EMDD-RL: FASTER SUBGOAL IDENTIFICATION WITH DIVERSE DENSITY IN REINFORCEMENT LEARNING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SAIM SUNEL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JANUARY 2021

Approval of the thesis:

EMDD-RL: FASTER SUBGOAL IDENTIFICATION WITH DIVERSE DENSITY IN REINFORCEMENT LEARNING

submitted by **SAİM SUNEL** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Prof. Dr. Faruk Polat
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Mehmet Tan
Computer Engineering, TOBB ETU _____

Prof. Dr. Faruk Polat
Computer Engineering, METU _____

Assist. Prof. Dr. Alper Demir
Computer Engineering, İzmir University of Economics _____

Date: 15.01.2021

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Saim Sunel

Signature :

ABSTRACT

EMDD-RL: FASTER SUBGOAL IDENTIFICATION WITH DIVERSE DENSITY IN REINFORCEMENT LEARNING

Sunel, Saim
M.S., Department of Computer Engineering
Supervisor: Prof. Dr. Faruk Polat

January 2021, 102 pages

Diverse Density (DD) algorithm is a well-known multiple instance learning method, also known to be effective to automatically identify sub-goals and improve Reinforcement Learning (RL). Expectation-Maximization Diverse Density (EMDD) improves DD in terms of both speed and accuracy. This study adapts EMDD to automatically identify subgoals for RL which is shown to perform significantly faster (3 to 10 times) than its predecessor, without sacrificing solution quality. The performance of the proposed method named EMDD-RL is empirically shown via extensive experimentation, together with the discussions on the effects of EMDD hyperparameters on the results.

Keywords: Subgoal Identification, Expectation-Maximization Diverse Density, Diverse Density, Reinforcement Learning

ÖZ

EMDD-RL: PEKİŞTİRMELİ ÖĞRENMEDE AYRI YOĞUNLUK YÖNTEMİYLE DAHA HIZLI ALT HEDEF BULMA

Sunel, Saim
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Faruk Polat

Ocak 2021 , 102 sayfa

Ayrı Yoğunluk (AY) algoritması, pekiştirmeli öğrenme alanında öğrenme sürecini iyileştirmek amacıyla otomatik olarak alt hedefleri bulma konusunda etkili olduğu gösterilmiş, çoklu-örnek öğrenme alanında öne sürülmüş ve bu alanda iyi bilinen bir yöntemdir. Aynı alanda önerilmiş olan Beklenti Yükseltmeli Ayrı Yoğunluk (BYAY) yöntemi, AY metodunu hem doğru sonuç üretme, hem de hızlı çalışma açısından geliştirmiştir. Bu çalışma pekiştirmeli öğrenme problemlerinde otomatik olarak alt hedeflerin tespiti için BYAY metodunu, algoritma üzerinde gerekli değişiklikleri uygulayarak, uyarlamaktadır. Uyarlanan yöntem, atası sayılabilecek olan AY algoritmasıyla tasarlanmış diğer algoritmadan tespit edilen alt hedeflerin kalitesini koruyarak 3 ile 10 kat daha hızlı çalışmaktadır. Tezde EMDD-RL ismiyle önerilmiş olan yöntemin sahip olduğu parametrelerin, algoritmanın performansı üzerindeki etkileri tartışılmış ve ayrıntılı deneylerle ortaya konulmuştur.

Anahtar Kelimeler: Alt Hedef Bulma, Beklenti Yükseltmeli Ayrı Yoğunluk, Ayrı Yo-

ğunluk, Pekiştirmeli Öğrenme

To my dear family and my darling

ACKNOWLEDGMENTS

First of all, I would like to thank my supervisor Prof. Dr. Faruk Polat for his generous support and leadership that guided me to the field that I am very grateful to be working in. Without his assistance, this thesis wouldn't have been in existence. Secondly, I would like to thank Dr. Erkin Çilden for his valuable comments and suggestions for solving the problems that I have faced during this work. Also, I would like to thank both Assist. Prof. Dr. Alper Demir and Hüseyin Aydın for their comments, discussions, conversations throughout the development of this study.

This thesis study is supported by 2210-A (2019/2) National Scholarship Programme for MSc students of the Scientific and Technological Research Council of Turkey (TÜBİTAK).

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

Reinforcement Learning (RL) deals with self-improving agents that learn to solve sequential decision problems solely based on a reward mechanism in a trial-error process. Depending on the rewards given, the agent tweaks its policy (decisions) and develops a behavior in such a way that it maximizes the accumulated reward in the long run. Reinforcement learning is mathematically formulated via Markov Decision Process (MDP). MDPs are represented via 4-tuple (S, A, R, T) where S is the set of states that the agent perceives, A is the set of actions that the agent can take, R is the reward function that is fed to the agent and informs the agent about benefit or harm of the applied action, and T is the transition function that models the transition probability of the agent moving from one state to another. In the course of learning, this framework governs the agent-environment interaction and the agent seeks to obtain optimal behavior. The main objective of the agent is to maximize the expected reward received ($\mathbb{E}[\sum_t \gamma^t R_t]$) over time, where γ is the discount factor that regulates the significance of the future rewards. Optimal behavior (optimal decision sequence, optimal policy) can be obtained by preferring the actions for each state in such a way that the accumulated reward is maximum. A well-known RL algorithm called Q-learning [2] can be employed for learning optimal behavior. Q-learning builds a table storing the expected value of each available action in every state and updates the table elements in accordance with rewards received by following the iterative rule:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R_t + \gamma \max_{b \in A} Q(s_{t+1}, b) - Q(s_t, a_t))$$

where α is learning rate, s_t is the current state of the agent, a_t is the current action taken by the agent, s_{t+1} is the next state that the agent has transitioned to after taking action a_t. After gathering sufficient number of updates, the agent can look up the table and decide which actions are preferable for attaining the maximum reward hence the optimal action sequence can be retrieved.

Options framework [3] extends the action semantic of the MDPs in such a way that the agent can leverage more complex actions instead of primitive actions. And the agent can carry out the learning procedure with both primitive actions and compound actions consisting of multiple primitive actions. Complex actions speed the learning procedure, hence the problem can be solved by spending less time [3]. The option is represented via 3-tuple $< I, \pi, \beta >$. I is the initiation set consisting of the states where the option is defined, π is the policy and β is the termination function of the option. These three components of the options need to be decided before employing this framework. This requirement has garnered attention to the research area of the methods that build options in the RL problems. The most important components of the options are both the termination function and the initiation set since the policy of the option can be learned via experience replay [4] after determining these two components. Further, the option termination function can be considered as the most important component since the initiation set needs to dwell around the terminating states (states whose termination functions output 1).

A single option may correspond to a subtask in an RL problem. The determination of the policy of the option in such a setting results in solving the subtask. For instance, in grid-world problems, the agent is supposed to navigate between rooms and arrive at the goal state. Obviously transitioning from one room to another may be regarded as a subtask of the bigger grid-world problem. An option that takes the agent to the other room sorts the subtask out for the agent. And the setting of the option could be as follows: the room states constitute the initiation set of the option and the termination function ceases the execution of the option at a gateway location.

Decomposing problems into smaller pieces and solving each subproblem deliver significant performance gain especially when the reward received by the agent is scarce [5, 6, 7, 8]. In grid-world problems, it is easier to grasp the benefit. When the agent is

not reinforced with frequent rewards, it spends its precious time wandering in several rooms, as time passes it eventually learns the optimal behavior. But if it is able to move to other rooms at the early stages of the learning, it can explore its environment swiftly, hence the agent's learning efficiency increases a lot (so does its chance of reaching the goal state earlier). The options framework equips the agent with this ability and facilitates quicker learning [3]. In the literature, most of the studies attempt to find these gateway locations in grid-world problems under the title of subgoal identification. More generally, the subgoal identification problem focuses on the critical states (subgoals) for the agent while reaching the goal state. As in the grid-world problems, reaching these states helps the agent to accomplish its goal and explore its environment quickly.

Different kinds of methods and techniques that attack the subgoal identification problem from different perspectives have been proposed. These methods can be categorized depending on the base methods that were employed. One group of studies make use of graphical algorithms and metrics where the transition history of the agent over states is represented as a graph. Over the graph, graph-based algorithms are run and metrics are calculated for subgoals [7, 6]. Another group devises metrics that work with state occurrences in the transition history of the agent [8, 9], while some others leverage classical machine learning techniques [10, 11].

Subgoal identification with Diverse Density [11] stands out among all others with its unique approach to the subgoal identification problem. It treats the problem as if it were a problem of the multi-instance learning paradigm. This machine learning paradigm was devised in the need for a solution to the drug activity prediction problem and is a generalization of conventional supervised learning. Within this paradigm individual data point labels are not known, instead, the data instances are grouped and only the label of the group is provided. Algorithms that tackle with this paradigm aim to perform classification, regression, clustering tasks with these grouped data instances. The algorithms might work at the data instance level or bag (group) level. The Diverse Density (DD) was the first multi-instance learning algorithm adapted for the subgoal identification problem. Episode trajectories of the agent are categorized as positive and negative bags depending on the trajectory's criteria of success (i.e. reaching the goal). The algorithm then finds the most critical state that is responsible

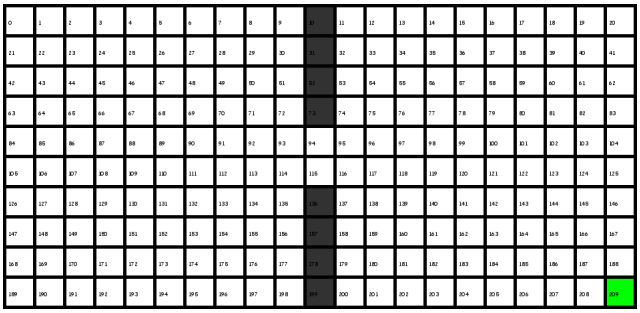
for the bag labels, which corresponds to the subgoal in the problem.

The data setting required by the multi-instance learning paradigm is compatible with the data gathered in RL agent experiences. The bags are the agent trajectories and the states are the instances in the bags. It is very natural to employ multi-instance learning algorithms for gathering useful information from the agent trajectories. For instance, Diverse Density's search for the most critical state aligns with the requirement of the subgoals pretty strongly. The subgoals are expected to be on the path to the goal state, present in many episodes that reach the goal [11]. Similarly, the DD expects the critical state to be present in positive bags and absent from the negative bags. The instance should reside in the trajectories that attain the goal state, not be present in unsuccessful trajectories (not ending with the goal state). In this study, we leverage another well-known multi-instance algorithm and adapt it to the subgoal identification problem in RL problems.

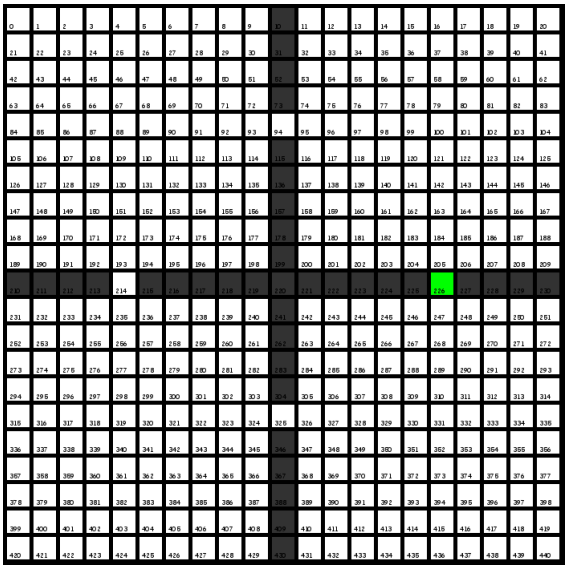
This thesis is mainly based on two studies: Subgoal Identification using Diverse Density (SDD) [5] and Expectation-Maximization Diverse Density (EMDD) [1]. The algorithmic decisions (such as considered bag counts, option creation procedure, etc.) and the experimental settings are inspired from the former work. The latter describes the algorithm that we have adapted for the subgoal identification problem. The EMDD algorithm formulated the most critical concept identification problem as an Expectation-Maximization procedure and in the multi-instance learning domain, it was shown to surpass the DD algorithm in both classification accuracy and running time. The EMDD was capable of pinpointing the most important concepts more accurately that were later used to label the unseen bags with less calculation overhead compared to the DD. Firstly it takes fewer instances into account during inner calculations, secondly, it uses a smaller instance group among which the critical concept is picked, whereas the DD involves every data instance in inner calculations and makes use of all instances that are in the positive bags for the concept identification. Its advantages over the DD in the multi-instance learning domain make the EMDD algorithm a strong candidate for subgoal identification, which is the main concern of this thesis.

In order to be compatible with the subgoal identification with DD study (SDD) [5],

we consider two grid-world problems for the subgoal identification throughout this study: two-room, four-room environments. Visualization of the environments are given in Figures 1.1a and 1.1b.



(a) Two-room Environment



(b) Four-room Environment

Figure 1.1: The grid-world problems that have been tackled in this thesis. The states of the environments are represented with numerical values. The green-colored states are the goal states. The agent cannot transition to black-colored obstacle states. When the agent attempts to move towards the obstacle states, it remains unmoved.

For both environments, the green-colored states represent the goal states. The agent can transition to white-colored states. The black-colored states are the obstacle states

through which the agent can not pass. If the agent takes an action that leads to an obstacle state, the agent remains unaffected (stays still). In the two-room environment problem (which consists of 210 states), the agent starts interacting with the environment in any state of the left room randomly. Its objective is to reach the goal state located at the bottom of the right room (depicted via green color) by passing through a gateway that splits the two rooms. Similarly, in the four-room (consists of 441 states) environment, the agent starts in any state of the left-hand rooms randomly and attempts to reach the goal state located between right-hand rooms (depicted with green color). Similar to SDD [5], in both problems, the agent is exposed to an action noise, which leads the agent to move in the intended direction with a chance of 90% (10% chance of moving in any direction).

1.2 Proposed Methods and Models

The EMDD algorithm was proposed depending on the needs of the multi-instance learning domain. We rule out its several steps and propose new techniques instead in order to adapt it and leverage it as a subgoal identification method for RL tasks. Firstly it requires multi-dimensional data to work with. Secondly, gradient ascent is employed to maximize probability values. Since we are working with grid-world problems, we need to eliminate the parts of the algorithm related to multi-dimensional data. Gradient ascent is a costly operation in terms of calculations. So we have dropped the gradient ascent steps and the maximizing instance is decided via individual score comparisons, as done similarly to SDD. The authors of the EMDD proposed alternative calculation techniques for some parts of the algorithm. We regard these as hyperparameters for our algorithm. As supported by our experiments, these alternative methods possess significant importance on the accuracy and speed performances of the algorithm. Another important component of the EMDD algorithm is that it picks its test instances (which will be used to obtain the critical state) from a relatively small group of data bags. We alter this selection strategy by considering all instances of the positive episode which has the fewest instances. Besides, we have also considered all positive bag instances for maximization similar to SDD. With elaborate experiments, we have observed the effects of these alternative methods and strategies

over both execution speed performance and subgoal identification accuracy. The experiment results show us that our adapted subgoal identification method significantly outperforms the SDD in terms of execution speed (3-10 times faster) by delivering either better or similar subgoal identification quality.

1.3 Contributions and Novelties

This thesis adapts a well-known algorithm of multi-instance learning for identifying subgoals in reinforcement learning problems. Some steps of the original algorithm are revised to conform to RL problems. Contributions of our thesis are as follows:

- Our algorithm surpasses its predecessor in both execution time and accuracy in identifying the subgoals in various test cases.
- Our algorithm is more resilient to large state-space sizes compared to its predecessor.
- Our algorithm identifies the subgoals with a fewer number of episodes in tested problems.

1.4 The Outline of the Thesis

This thesis attempts to attack the well-known subgoal identification problem in reinforcement learning tasks. We start with introducing reinforcement learning (RL) and touching upon a well-known algorithm for solving RL problems. Then we make a brief introduction about the options framework that enables RL algorithms to work with abstract level actions. Before delving into some important subgoal methods in the literature, we continue with the multi-instance learning paradigm, its characteristics, algorithms used since later the ideas and assumptions made in multi-instance learning are crucial for the related work section. Then we present the previous important studies that tackled the subgoal identification problem in the literature. We emphasize the study that made use of the multi-instance learning paradigm for this purpose since our work considers it as a base study. The detailed algorithm descrip-

tion of the Expectation-Maximization Diverse Density is given also in that section. After introducing all the necessary information, we present our algorithm EMDD-RL by explaining the assumptions that it has made and giving the reasons why it has been developed in that way in an elaborate manner. We then present experimental setups and the results of the experiments conducted over the EMDD-RL and its predecessor (SDD) to show how better the EMDD-RL performs against the other algorithm in both speed and subgoal identification accuracy performances. The results and the hyperparameters of the EMDD-RL are elaborated in that section as well. Finally, in the last two chapters, we discuss our method's impact, advantages, disadvantages, possible usage areas in other fields and conclude our thesis.

CHAPTER 2

REINFORCEMENT LEARNING

2.1 Reinforcement Learning Paradigm

Reinforcement learning constitutes the third paradigm of the machine learning field after supervised and unsupervised learning. The main objective of the paradigm is to realize the learning on a trial and error basis. Different from the other learning types, the problems of this paradigm require sequential decision making for the solution, where multiple decisions must be made and the previous decisions affect the outcomes of the future decisions directly. The programs (agents) are reinforced depending on how they have behaved on a particular problem. The feedback signals may be immediate or delayed. With those feedback signals, the agent improves its behavior (decision, policy) in order to attain the goal objective of the problem.

In a more elaborate manner, the program interacts with the problem. It applies its decision on the problem and observes the feedback signal. By repeating this interaction procedure many times, the agent (program) explores the problem domain and develops the optimal decision sequence (optimal policy) in order to be able to solve the problem. The problem to be solved must be adapted to conform to the needs of this interaction procedure. It should be able to provide the required information for the agent to learn.

The feedback signal can be defined to be a reward or cost for the decision. As there are objection functions that are minimized or maximized in other paradigms, the objective function in reinforcement learning is to minimize expected accumulated loss or maximize expected accumulated reward depending on the definition of the signal. Since the signal received changes during decision making, it is threaded as a proba-

bility variable hence its average on a long period is considered. By considering the maximum or minimum of the accumulated feedback signal, in fact, the program seeks the "best" decision sequence (optimal policy) since the decision determines the value of the reward or the cost.

Chess could be regarded as a good problem for the reinforcement learning paradigm. It requires multiple decisions to be made one after another in order to defeat the opponent. Each opponent assesses the current snapshot of the board and decides which action to take. The current action taken is totally dependent on previous ones since all previous decisions have led to the current situation of the board. An optimal policy, sequential moves that result in defeating the opponent, requires to have made the optimal move in every turn. A reinforcement learning agent that tackles Chess needs to learn the optimal decision sequence for every possible game configuration. The reinforcement signal for the agent could be loss or win status at the end of the game.

2.2 Markov Decision Process

As other paradigms are based on mathematics, reinforcement learning is also built upon a mathematical framework called Markov Decision Process (MDP) [12]. This framework provides a mathematical description of the reinforcement learning problems. MDP is a 4-tuple (S, A, R, T) where S is the set of states in which the agent can reside, A is the set of available actions from which the agent can pick and apply in a state, $T : S \times A \times S \rightarrow [0, 1]$ is the transition function that governs the transition probability from one state to another after a particular action is taken, $R : S \times A \rightarrow \mathbb{R}$ is the reward function that outputs a numerical value indicating the benefit or loss amount of the action taken in a particular state. The Markovian property stems from the fact that the next state and reward value are determined by only the most recent state and the action taken rather than the history of all state-action pairs (all past information is infused in the current state.). Hence the agent makes a decision depending on the current state. The agent observes its current state information (in Chess example, the current status of the board) provided by the MDP and applies an available action on the current state (a legal move for the current board state). The action applied leads

to a state transition on MDP (the current status of the board changes with a piece moving to a new location), a reward value is obtained via the reward function (e.g. at the end of the game, the agent is given a reward of +1 when it wins otherwise -1). The agent receives the new state and reward information and realizes its learning task by perpetuating interaction with the MDP. When the agent reaches its goal, it starts from the beginning (when the current game is complete, a new game is initiated with all pieces on their starting locations). The interaction of the agent between the initial starting state and ending state of the MDP is coined as an episode. Episode history consists of the current state, next state, current action, currently received reward at each step of the agent.

The agent aims to maximize the accumulated reward on the episode. Since the actions taken are different from episode to episode during the learning stage, the reward received is threaded as a probability variable and its average accumulated value is sought over many episodes. More formally ($\mathbb{E}[\sum_t \gamma^t R_t]$) is maximized over time, where γ is the discounting factor. It governs the effect of the possible future rewards at the current time step. When it is picked to be close to 1, the agent takes the future reward into consideration during learning, on the other hand, if it is close to 0 the future rewards are discarded and the agent solely depends on the immediate rewards received. The action sequence that delivers the highest expected accumulated reward is the optimal policy of the agent. After extracting the optimal policy during learning, the agent can apply this policy to every new episode of the problem.

The reward value provided can be immediate or delayed (can be regarded as a special case of immediate where cost or reward of 0 is given). With the immediate reward function, the agent is provided with a numerical value at every MDP step. Delayed rewards are provided after a specific number of steps or at the end of the episode (as given in the Chess game). The agent can discern which action has resulted in loss or benefit via immediate feedback whereas it becomes quite difficult to find out which intermediate action or actions have been responsible for the success or the loss via scarce reward. This problem is coined as credit assignment problem and the learning mechanism for the agent should be able to obtain the required policy even on the tasks that exhibit this type of problem.

Since the MDP's state and action components are sets, they can be continuous or discrete. Besides, the individual elements of these sets could be multi-dimensional as well. In a continuous set, each element has a continuous numerical value and the size of the set is infinite whereas in the discrete case, elements are assigned to one of the finite values. Multidimensional element has more than one component and can be represented via vector notation. As a good example of continuous multidimensional state space, Mountain Car (Moore90) problem can be given. The state of the problem is represented via 2-D continuous components: location of the car (range of -1.2 to 0.6), speed of the car (range of -0.07 to 0.07). For discrete multidimensional state space, the Chess game could be considered. The board can be represented via a 64-dimensional vector each component of which stores the piece information (which piece exists on this location, finite values) on the corresponding location. As regards the action set, controlling the 6-DOF (degrees of freedom) robot arm requires a 6-dimensional continuous action vector. Values can be the numerical force amount to be exerted on each actuator of the arm. Similarly, a stock market problem might require a multidimensional discrete action vector where each dimension corresponds to a specific share and the value on this dimension specifies the sell or buy transaction.

Throughout this thesis, we consider MDPs with finite, 1-dimensional discrete state and action sets.

2.2.1 Optimal Policy Learning

All learning methods for the MDP basically leverage the accumulated expected reward/cost sum and minimize or maximize the value. In order to demonstrate the relation between the reward/cost and the policy better, we need to elaborate the expected sum with policy and initial state since the MDP starts with an initial state and follows a sequence of actions (policy). We define accumulated sum as $J_{\pi_{0\rightarrow T}}(s_0) = \mathbb{E}[\sum_t^T \gamma^t c(s_t, a_t)]$ where $c(s_i, a_i)$ the reward/cost incurred due to the action taken by a_t in state s_t, π is the policy ($\pi = \{a_0, a_1, a_2, a_3...\}$) and the state transition sequence of the MDP is $s_0, s_1, s_2, s_3, ...$. Further we can decompose the J as follows: $J_{\pi_{0\rightarrow T}}(s_0) = \mathbb{E}[c(s_0, a_0)|s_0] + \gamma\mathbb{E}[\sum_{t=1} \gamma^{t-1} c(s_t, a_t)|s_0]$, equivalently $J_{\pi_{0\rightarrow T}}(s_0) = \mathbb{E}[c(s_0, a_0)|s_0] + \gamma\mathbb{E}[J_{\pi_{1\rightarrow T}}(s_1)|s_0]$. When we keep expanding the cost value, we can

observe a general recursive dependence on cost/reward values $J_{\pi_{t\to T}}(s_t) = \mathbb{E}[c(s_t, a_t)] + \gamma\mathbb{E}[J_{\pi_{t+1\to T}}(s_{t+1})|s_t]$. Hence whole decision sequence can be split and examined individually and the cost/reward value can be calculated recursively. It can easily be recognized that by calculating the cost in this way, in fact we keep track of the state's individual costs. The J function stores the expected future cost of the states.

Intuitively, in order to attain the optimal policy, we might pick the action that incurs the least amount of the expected cost in every step of the decision making. This strategy was shown to be optimal and to deliver the least accumulated cost irrespective of the initial starting state (Bellman optimality principle) [13]. Formally, it can be represented as $J_{\pi_{t\to T}}(s_t) = \min_{\mathrm{a}_t}\{c(s_t, a_t = \mathrm{a}_t) + \gamma\mathbb{E}[J_{\pi_{t+1\to T}}(s_{t+1})|s_t]\}$. The first part of the expression is the received reward and the second part is the expected cost value of the next state. we can expand the second term as $E[J_{\pi_{t+1}}(s_{t+1})|s_t] = \sum_x J_{\pi_{t+1\to T}}(s_{t+1} = x)P(s_{t+1} = x|s_t)$, where $P(s_{t+1}|s_t)$ is the probability of one of the next states for the given current state s_t. From the definition of the MDP we know that probability of next state depends on the current state and the action and given by the transfer function $T(s_{t+1}|s_t, a_t)$. After placing and performing the required probability calculations we get $E[J_{\pi_{t+1}}(s_{t+1})|s_t] = \sum_{\mathrm{a}}\sum_x J_{\pi_{t+1\to T}}(s_{t+1} = x)T(s_{t+1} = x|s_t, a_t = \mathrm{a})P(s_t, a_t = \mathrm{a})$. Since we seek the action incurring the least amount of cost, in the expectation calculation step we need to look a single action. Hence overall update rule turns out to be:

$$J_{\pi_{t\to T}}(s_t) = \min_{\mathrm{a}_t}\{c(s_t, a_t = \mathrm{a}_t) + \gamma\sum_x J_{\pi_{t+1\to T}}(s_{t+1} = x)T(s_{t+1} = x|s_t, a_t = \mathrm{a}_t)P(a_t = \mathrm{a}_t|s_t)\}$$

This equation is coined as Bellman optimality equation. It consists of a system of equations and the solution of the system yields the optimal policy for the MDP.

Depending on the information available to the MDP solver (agent), solution methods can be categorized into two: model-based, model-free methods. If the agent is granted with the state transition and reward functions directly, it can employ Value Iteration or Policy Iteration algorithms on the given MDP. Since the agent knows all information contained in MDP it can solve the MDP without interacting with it (offline learning). Both methods employ the Bellman optimality equation and approach it from similar perspectives. Their convergence guarantees are provided as well.

In the case that the agent lacks the transition and the reward function knowledge, three

other algorithms that approximate the Value-Iteration and Policy Iteration algorithms were proposed: Temporal difference learning [14], Q-learning [2], Sarsa Learning [15]. These algorithms gather information by interacting with the MDP (online learning). Convergence properties of these algorithms are guaranteed on the condition that a sufficient number of interactions with the MDP are provided. Transition functions of most of the real-world problems can not be known in advance. Hence these algorithms are well suited for real-world problems since they do not involve the transition function directly in their calculations.

All methods can be gathered under the name of Dynamic Programming (Bellman) methods since each method divides the whole problem into sub-problems and attempt to solve the whole problem by dealing with sub-problems, which is thanks to the recursive nature of the Bellman optimality equation.

The Policy Iteration algorithm consists of two main stages: policy evaluation, policy improvement. The algorithm forms a policy randomly by picking one action available in every state. Then the algorithm calculates the accumulated cost value of every state with its current policy. After obtaining the cost values, in its second stage, the algorithm amends the current policy in a such way that the updated policy attains better cost value. Then these two stages are followed till the current policy doesn't change any longer. The policy evaluation is implemented via the update rule:

$$J_{\pi_{t\to T}}(s_t) = c(s_t, a_t) + \gamma \sum_x J_{\pi_{t+1}\to T}(s_{t+1} = x) T(s_{t+1} = x | s_t, a_t) P(a_t | s_t)$$

The improvement of the current policy is implemented as:

$$\pi_t = \arg\min_{\mathrm{a}_t} \{c(s_t, a_t = \mathrm{a}_t) + \gamma \textstyle\sum_x J_{\pi_{t+1}\to T}(s_{t+1} = x) T(s_{t+1} = x | s_t, a_t = \mathrm{a}_t) P(a_t = \mathrm{a}_t | s_t)\}$$

The Value Iteration algorithm fuses the two stages of the Policy Iteration into one. While calculating the state cost values, the algorithm considers the best action on every state. The value update operation needs to be repeated till the difference between previous and current state values drops below a predefined threshold value. The update rule of the algorithm is the same as the Bellman optimality equation, the policy can be extracted similar to the policy improvement step of the Policy Iteration or the agent can allocate a separate memory for the policy and keep changes on this memory without consulting the values of states.

Temporal Difference learning, on the other hand, totally depends on the interaction with MDP. Each action yields a new sample for learning. By obtaining sufficiently many samples we can approximate $J_{\pi_{t\to T}}(s_t) = \mathbb{E}[c(s_t, a_t)|s_t] + \gamma\mathbb{E}[J_{\pi_{t+1\to T}}(s_{t+1})|s_t]$ with Robbins-Monro stochastic approximation [16] which delivers us the following update rule:

$$J^{new}_{\pi_{t\to T}}(s_t) = (1-\alpha)J^{current}_{\pi_{t\to T}}(s_t) + \alpha(c(s_t, a_t) + \gamma J_{\pi_{t+1\to T}}(s_{t+1}))$$

where α is the learning rate for the update amount, $J^{current}$ is the current value of the state s_t and J^{new} is the state's new updated value.

Similar to Temporal Difference learning, Q-learning and Sarsa learning keep track of the state value via samples emerging from the interaction with the MDP. Different from the all previous method these two methods stores the value of the state-action pair instead of the only state value. The advantage of keeping the state-action value over the state value is that the policy can easily be extracted by only looking at the value of the action on the state. In order to extract policy from the state values, we need to perform the policy improvement calculation of the Policy Iteration which is more computationally costly compared to a single table lookup operation. The relation between the state value and the state-action value can easily be shown via $J(s_t) = \min_a Q(s_t, a)$ since the lowest state value can only be attained when the action that leads to the smallest cost is preferred. By simply modifying $J(s_t)$ with action information we can obtain $Q(s_t, a_t) = E[c(s_t, a_t)|s_t, a_t] + \gamma\mathbb{E}[J_{\pi_{t+1\to T}}(s_{t+1})|s_t, a_t]$. We can replace $J(s_{t+1})$ via its corresponding Q value function, $Q(s_t, a_t) = E[c(s_t, a_t)|s_t, a_t] + \gamma\mathbb{E}[\min_{\mathrm{a}_{t+1}} Q(s_{t+1}, a_{t+1} = \mathrm{a}_{t+1})|s_t, a_t]$. Again, by leveraging Robbins-Monro stochastic approximation, the following update rule for Q-learning can be obtained:

$$Q^{new}_{\pi_{t\to T}}(s_t, a_t) = (1-\alpha)Q^{current}_{\pi_{t\to T}}(s_t, a_t) + \alpha(c(s_t, a_t) + \gamma \min_{\mathrm{a}_{t+1}} Q_{\pi_{t+1\to T}}(s_{t+1}, a_{t+1} = \mathrm{a}_{t+1}))$$

Similar to TD, $Q^{current}$ is the current value of the state-action pair and Q^{new} is the new updated value. The minimum (equivalently maximum) valued next state-action pair is considered during the update. During learning, in each step, the agent may not take the maximum valued action in the next state due to the action selection strategy mentioned in the next paragraph. The maximum action selection introduces a bias to learning and hinders learning in some problems [17, 15]. Sarsa (State, Action,

Reward, State, Action) [15] learning on the other hand eliminates the maximum-valued action selection and instead samples a new action on the next state. When we modify the Q-learning update accordingly, we get the following update rule for Sarsa:

$$Q^{new}_{\pi_{t\to T}}(s_t, a_t) = (1-\alpha)Q^{current}_{\pi_{t\to T}}(s_t, a_t) + \alpha(c(s_t, a_t) + \gamma Q^{current}_{\pi_{t+1\to T}}(s_{t+1}, a_{t+1}))$$

In the next step of the update, a_{t+1} becomes a_t. In the Q-learning update, the value of the current action-state is updated via some other action that may not get taken in the next step. On the other hand, Sarsa updates the current value with the action taken in the next step. This fundamental discrimination between these algorithms leads to the definitions of on-policy and off-policy learning since all taken actions (on-policy) are being used to update the value function in Sarsa learning whereas a maximum valued action that may not get taken (off-policy) in the next state is used to update in Q-learning. More concisely, on-policy methods update the value function with their actual policy distribution, on the other hand, off-policy methods can follow some other policy distribution for learning.

Different from the model-based methods, model-free methods have to contain some randomization during learning. They can not rely on their initial random policy since if there is no randomization in their interaction they get stuck in repeating the same actions over and over again. No learning can take place so these algorithms have to have a randomization component in choosing their policies in the course of learning. On the other hand, after acquiring a sufficient amount of experience and developing some level of behavior, the effect of this random component should be lowered and the algorithm should depend on the learned behavior more. The randomization component equips the learning algorithm with an exploration ability hence the agent can decide to explore or decide to rely on what has been learned so far (exploitation). The trade-off between exploration and exploitation is one of the main problems in the RL field. The intensity of the randomization (exploration) should be at such a level that it should not degrade the learning performance due to being too much random and it should enable the algorithms to acquire the expected behavior. At the early stages of the learning, the randomization can be kept high but as the learning progresses the level of the randomization can be declined but the intensity of the randomization should be decided and adjusted carefully depending on the problem at hand. Temporal Difference learning, Q-learning and Sarsa learning incorporate the exploration

component in their policy selection procedure. With some certain probability k they choose the action from learned policy and with $1 - k$ probability they take a random action.

With the enhancements of the computational devices, the "deep learning" field has been able to thrive significantly in the last decade. From medical applications to image synthesis, it has prevailed in a vast majority of the research areas. In the cases where the tasks to be tackled demand continuous-valued state representation or continuous action, the algorithms we have mentioned above have to meet these requirements. Unfortunately, the original algorithms store the value functions in a tabular data structure. For some problems keeping a tabular memory may not be feasible even if the task can exhibit a discrete nature (e.g. Chess, for each state configuration the memory demand is impractical.). In the literature to circumvent these difficulties, function approximators or discretization methods have been employed [18, 19, 20]. The problem with the discretization procedure is that it might cause information loss. Different from their tabular versions, when these algorithms are combined with a function approximation technique, their convergence properties diminish theoretically, hence they might converge or diverge [21]. Neural networks are very strong function approximators [22]. Hence the value functions that are stored in a tabular data structure in classical reinforcement learning can be approximated via neural networks easily. The first successful ground-breaking study was Deep-Q Learning [23]. The neural network in this study learns the action-state value function and the loss function that is minimized is the square of the Q-learning update rule mentioned above (to yield a smoother gradient vector for the back-propagation algorithm). The states are represented via images of Atari game screenshots and the actions are the keys pressed on the Atari controller. The proposed architecture was able to exhibit better performance in 49 Atari games compared to human players. Although function approximation undermines the convergence properties of the algorithms mentioned, it enables us to directly learn the policy without consulting any value function learning. If the function approximator is differential (e.g. the neural networks are) with the help of policy gradient theorem [24], we can directly learn the policy function without suffering divergence problems since its main learning infrastructure is gradient descent or ascent algorithm (REINFORCE [25]). Pure policy gradient algorithm

may severely suffer from variance in gradient updates, a value called baseline can be subtracted in order to alleviate the variance. Besides policy function, the baseline value function can be learned together and this combination leads the actor-critic architectures [26, 27, 28]. Each purposed method has its advantages and disadvantages and none of them is superior to the others for all problems [29], which is natural to expect thanks to the No-Free Lunch theorem.

So far we have tried to convey a general basic knowledge for the reinforcement learning field. For more elaborate treatment and more information for the field can be found in [30, 31]. The ideas that emerged in the classical tabular reinforcement learning has affected and directed deep reinforcement learning, we believe it will continue to do so in the future.

2.3 Options Framework

Options framework [3] extends the classical MDP model with multi-step action semantic. As we have introduced MDPs in the previous section, MDP advances from one state to another by taking the action and receiving a reward in one time step. Semi-Markov Decision Process (SMDP) is another important mathematical model that incorporates time-variable transitions as an extension to MDP. Basically, the semi-MDP transitions to the next state after waiting for a varying time interval. On one side only a single time step is required to transition to the next state on the other side transition occurs after a varying time step. The options framework fuses and adopts these two different cases in such a way that the proposed formulation improves the learning capabilities and attains better performance for reinforcement learning problems. Different from the SMDP, the options framework discretizes the continuous-time and manipulates the actions in discrete time intervals. In other words, the options framework combines the extra waiting time of the SMDP with actions and the actions (policy) of the option can be changed and learned. Formally option structure is defined via a tuple: $< I, \pi, \beta >$. I corresponds to the set of the states over which the option is defined. β function yields the probability of the option termination in a particular state and π corresponds to the policy of the option. When the option is executed, it follows its policy until it terminates. It consults its policy

function in each time step and checks the termination function after taking one step. If the termination function decides to halt, the option is dismissed, and if defined a new option is run.

The options framework broadens the semantic of the action in the MDP. The actions in the MDP framework can easily be thought of as a special case of the options where each action is a single option, each action is defined over all states and lasts only a single time step. Depending on the termination function, the duration of the option is determined hence the length of the actions can vary. The policy of the option can exhibit more complex behavior compared to the action semantic of the MDP. Hence, abstraction over the action semantic can be acquired. Since the primitive actions (actions of the MDP) can be considered as options as well, the options framework facilitates learning with both primitive actions and complex behaviors at the same level of treatment. Similar to MDP, learning methods for Q and V functions were provided. When the state transition function (model) is not known in advance, as done in MDP these functions can be approximated. The update rule for Q learning version of the option (SMDP Q-Learning) [32] is as follows:

$$Q(s_t, o_t) = (1-\alpha)Q(s_t, o_t) + \alpha(c_l + \gamma^l \max_{o_{t+l+1} \in O(s_{t+l+1})} Q(s_{t+l+1}, o_{t+l+1}))$$

where Q stores the expected cost value of taking option o_t in state s_t. l is the step count of the option, c_l is the accumulated reward received by applying the option. The Q values are updated after the option terminates since c_l value must be calculated. As in the Q-learning, the maximum/minimum valued option for the selection (in state s_{t+l+1}) is considered.

Waiting till the option termination poses a severe restriction for the option. If option execution takes too much of the time, it will severely slow down learning, and dismissing the option before it terminates could result in better performance [3]. Thus, interrupting options were introduced in order to make the options more flexible. Basically, if a better-valued option exists for a particular state, the current option can be abandoned. Since the definition information of the option can be retrieved from the initiation set of the options, we can consult the Q function for a particular state to find out the better option that incurs a higher/lower cost value if employed.

Inspecting and manipulating the option policy at the step level (discretized time inter-

vals) give rise to step-wise value learning methods for the options. In the SMDP-Q Learning update, cl needs to be calculated. Options need to be run many times for reliable SMDP-Q updates. Since the actions are taken in a step-wise manner, the immediately received rewards can be leveraged to update more than one option value instead of waiting for the option to terminate. Hence, if two or more options share similar policies over a set of states, a reward value received by following a particular option can be used to update the value of the others without executing them. Such a strategy also provides learning efficiency since the option update does not have to be waited till it terminates and more than one options values can be calculated via a single step reward. This type of value learning was coined as one-step intra-option learning [33] and performs the following update rule:

$$Q(s_t, o_t) = (1 - \alpha)Q(s_t, o_t) + \alpha(c_{t+1} + \gamma U(s_{t+1}, o_t))$$

where c_{t+1} is the immediate reward after applying the option policy (action) in state s_t for one step and $U(s_t, o_t) = (1 - \beta(s_t))Q(s_t, o_t) + \beta(s_t) \max_{o \in O(s_t)} Q(s_t, o)$. This update rule is performed for every option that is compatible with the applied action.

Besides performing learning, another critical point for the options is how the option should be formed. In the literature, depending on the problem types (grid-world, continuous, etc.) many a method has been proposed [5, 7, 34, 6]. General idea is that in the problems, some groups of states are critical for attaining the goal, if the learning algorithm manages to reach these critical states before the subgoal, the learning problem can be simplified (learning can speed). Options can be formed in the vicinity of those critical states. Over a region, the initiation sets of the options can be defined. The policy of the option can be obtained by inspecting the agent's behavior history in such a way that it helps the agent to reach those critical states (e.g. experience replay [4]). The termination function can halt the option when the option reaches one of the critical states.

The method that we propose in this thesis can also be used to form options since its main purpose is to identify the critical states for attaining the goal. In section 5.2.8, we have employed our method to form an option that delivers a faster learning performance compared to a Q-learning agent.

CHAPTER 3

RELATED WORK

3.1 Multi-instance Learning

Multi-instance learning (MIL) paradigm emerged in the need of predicting the molecules whether they were capable of forming compounds in the drug discovery problem [35]. The main problem was that for a particular molecule there were many molecular structures (conformation) to test its conformity for the target chemical reaction and all possible structures couldn't be tested in a laboratory environment due to the costs of the processes. If somehow we could assert the conformation of a new (untested) molecule for the target compound we would save time and afford by inspecting and learning from the molecules that were known to form the target compound. For a particular molecule that involves binding, there might be more than one conformations that allow the binding. When we consider the classical supervised learning setting for this problem, at first glance we realize that it is not suitable. If we directly feed the conformations of a particular molecule to the learning system, we need to label both binding and non-binding conformations to the tag of the molecule. Since both suitable and non-suitable conformations would be labeled with the same tag, classical supervised algorithms wouldn't work.

Basically, only the label of the molecule is provided, tags of individual conformations are not known. In multi-instance learning terminology, a group of the instances is called a bag, the molecules are bags where their conformations are their instances and only the labels of bags are known. As a second example, let's treat the problem of testing whether a text document is a scientific document or not. Assume that we have a sufficient number of text documents where some documents are known to be

scientific and the remaining ones are non-scientific. Each text document consists of some words related to and unrelated to science. In multi-instance learning setting each document corresponds to the bags and the instances of the bags are the words. Only the types (labels, scientific or non-scientific) of the documents are known. Although the document consists of many words, only one or some of the words might be responsible for the label of the document. The presence of scientific words and non-scientific words in the same text document poses a serious ambiguity for learning. Learning algorithms have to cope with this ambiguity and be able to classify the unseen bags with sufficient accuracy.

The classical supervised learning where each instance has its label is a special case of the multi-instance learning paradigm. Each instance corresponds to a distinct bag and each bag contains only a single instance. The difficulty arises when the bags contain more than one instance. The instances that contribute to the label of the bag might reside with the other unrelated instances for the label in the same bag. Besides, different bags might have different sizes with respect to instance count. Different algorithms that approach the problem from different aspects of the data have been proposed. Some algorithms work at the instance level and some others at the bag level. For instance level algorithms, the main objective is to identify those critical instances and make use of them for the task at hand. If the task requires predicting the label of an unseen bag then these critical instances are employed to validate. Some algorithms directly obtain the necessary information at the bag level without delving into the instance level and use this information for the task. Some other techniques convert the multi-instance learning problem into a classical supervised learning setting by applying some sort of transformation and embedding. Hence the usual supervised algorithms (neural networks, SVM, decision trees, etc.) become available for the multi learning paradigm as well.

Formally the multi-instance learning paradigm problems consist of bags represented via $B_i = \{B_{i,1}, B_{i,2}, B_{i,3}, ...\}$ where each $B_{i,j}$ represents a single instance of the bag B_i. Each bag has its label L_i and the dataset of the MIL consists of $< B_i, L_i >$ pairs. The objective of the learner of the MIL is to approximate a function defined from bag space to label of the bag ($h : N^X \rightarrow L$ where X is instance space, $N^X = \{B : X \rightarrow N\}$, $B(x)$ is the count of the instance x in $B \in N^X$ and $x \in X$ [36]) as close as

possible. Besides classification problems (in the early works of the MIL, dichotomies (two label classification) were considered but classification further has been extended to multi-class labels as well.), regression, clustering problems have been attacked in the multi-instance learning domain. In regression problems, bags are tagged with numerical values instead of class labels. Clustering on the other hand requires no labeling and groups the bags by measuring similarity/dissimilarity metrics.

Different assumptions about how the individual instances affect the label of the bag have been proposed in the literature (here we consider dichotomy classification, positive and negative bags). It is obvious that the instances decide the label of the bag but what kind of relations there are between the instances and the bag is an open question that needs to be answered in advance. The standard assumption which was employed in the earlier works of multi-instance learning paradigm [35, 11] presumes that each instance has its own hidden label, the label of the bag is decided by ORing the individual instance labels ($l(B_i) = l(B_{i,1}) \vee l(B_{i,2}) \vee l(B_{i,3}) \vee ...$). In addition to the standard assumption, other count-based (Weidmann et al.) and probabilistic (weighted, product rule) relations have been employed. On the other hand, the soft bag assumption further loosens the constraint that negative bags should not contain positive instances (previous assumptions are based on this). The assumptions are considered depending on the problem and their superiority can not be claimed on every multi-instance problem [37].

Instance-based methods operate at the instance-level and attempt to label individual instances. Depending on the considered assumptions, the label of the bags is obtained. Diverse Density (DD) algorithm [11] measures a probabilistic value (Diverse Density) to assess the importance of the individual instance on the label of its bag. The most influential instance is then employed to classify new unseen bags. Similarly, the Expectation-Maximization Diverse Density (EMDD) [1] algorithm considers DD's searching for the critical instance as an Expectation-Maximization problem and identifies the critical instances more accurately and faster compared to DD. Logistic regression [38] models the individual instance probabilities via logistic function and calculates the bag probabilities via normalized geometric mean of instance probabilities. The algorithms that convert the multi-instance learning problem into a classical supervised learning setting employ instance transformation. Once converted, any

well-known supervised technique could be used. MILES [39] constructs a feature vector by considering a probabilistic similarity between each instance and bags and SVM is used for the classification of instances and bags. Similarly, MILD [40] builds a distance-vector where each feature corresponds to the distance between the instance and a particular bag (for each instance such a vector is constructed). Then SVM can be used to label both instances and bags by following the standard assumption. On the other hand, algorithms that work at the bag level attempts to solve the problems by considering similarity/distance metrics on bags. Similar to the instance level case, methods that translate the MIL problem to supervised learning directly transforming the bags have been proposed. CitationKNN [41] study employs the K-nearest neighbor algorithm (via Bayesian and citation selection mechanisms) by measuring the bag distances via a modified Hausdorff distance. The MILD study also provides a feature vector construction technique for supervised learning where each feature of the vector corresponds to the distance between the bag and instances (for each bag a vector is constructed). Again SVM method can be used for classification.

In this thesis, we are adapting the Expectation-Maximization Diverse Density (EMDD) algorithm for the subgoal identification problem in RL. In the next sections, we present in an elaborate manner how the other well-known multi-instance learning algorithm (DD) was used for this purpose previously and introduce our adapted algorithm from the EMDD algorithm.

For a more elaborate treatment of the multi-instance learning, [37] may be referred.

3.2 Related Work

In this section, we represent previous important methods that tackled the problem of subgoal identification problem in RL problems. Since our study is based on multi-instance learning, we treat multi-instance learning more elaborately. Our subgoal identification method heavily depends on the previous work that attacked this problem with a different multi-instance learning algorithm. We also emphasize this work and explain how our method is derived in the following section.

Options framework [3] extended the notion of action in MDPs and facilitate learning

policy at a more abstract level. In MDPs, the agent carries out the learning at the primitive action level (single actions). With the options framework, the agent is enabled to use the actions that consist of multiple primitive actions and last multiple steps. Hence the agent can learn both at the primitive level and at an abstract level. The option study didn't attack the subgoal identification problem while demonstrating the method's usefulness and assumed that the subgoals were given in advance. This garnered the attention to subgoal identification research. It is because once the subgoals are identified useful options can be created by considering the subgoals. Hence the problem can be solved faster [3].

With the fostering of options framework, studies that seek the subgoal/bottleneck locations have emerged to be able to divide the original problem into smaller problems and these smaller problems are solved by making use of the options. Most of the studies focused their attention on grid-world (we also tackle these types of problems in this thesis, Figure 1.1) reinforcement learning problems because the subgoal/bottleneck locations are more apparent to the eye. In grid-world problems, the agent has to avoid obstacles and navigate through many rooms to reach the goal. The gateways enable the agent to transition to another room, if the agent cannot pass through those gates, it has to navigate through the room where it is. So the gateways areas are special to the agent. When it succeeds in passing through a gateway, it encounters a new room, it enlarges its space information hence it can learn more about the problem. Also in this way, the agent explores its environment and may find multiple alternative ways to attain the goal. Otherwise, it has to spend most of its valuable time getting stuck in a single room, and nothing much about the problem can be learned. Therefore the main objective of the subgoal identification for these types of RL problems is to pinpoint the gateway regions (locations). If at the early stages of the learning, the agent develops option/options that facilitate itself to navigate through other rooms, it can both explore its problem more efficiently (quickly) and attain the goal faster. Such an ability will be very beneficial especially when the agent is provided scarce rewards. Normally the reward is the driving force for the agent to explore and learn, but with option creation, the agent will still have the ability to explore its environment even if the agent lacks the valuable reward signal.

In the literature different approaches to identify the subgoal regions have been pro-

posed by approaching the problem from different perspectives. One method regards this problem as a multi-instance learning problem (as our method in this thesis). A group of work models the environment as a graph and graphical metrics or methods are employed. Another group focuses on count-based metrics. Each method has advantages and disadvantages depending on the problem. When the space size of the problems is too large, the execution of the graphical methods becomes cumbersome while finding the solution. Some methods identify a global property in whole graphs, which renders finding other subgoals difficult. For instance, one study considers the Max-Flow algorithm. Since the bottleneck node is searched over whole the graph, after identifying the first one, the question of how to identify the second subgoal arises. Since the graph may not be partitioned into smaller subgraphs. Besides, the stopping criteria for the partition needs to be determined.

When the agent traverses the environment, it transitions from one state to another. This journey can be represented with a graph. Nodes are the states over which the agent has passed and the edges are created between two nodes if the agent has transitioned from one to the other. When virtually displayed the constructed graph looks almost identical to the photographic representation of the environment. When we match these two visuals, the gateway locations (the subgoal regions) correspond to the border nodes that connect highly connected subgraphs (rooms). These border nodes can be identified by graph metrics or algorithms that accentuate these nodes. Q-cut study [7] considers the transitions of the agent as a flow on the graph. The nodes that throttle the flow corresponds to the locations of the gateways in the environment since these nodes attach highly connected graphs together hence the flow is restrained by them. Authors employ Max-Flow/Min-Cut to find bottleneck locations (subgoals). After identifying the cut, a threshold value is used to assess the quality of it. If the cut is good enough as a subgoal, options are created by making use of the agent's transition history. In order to solve multi-subgoal identification (as mentioned above), the authors propose Segmented Q-Cut which performs the operations with Q-Cut but interactively. As the agent traverses states, it constructs its transition graph and Min-Cut is calculated repetitively. If the cut quality metric exceeds a threshold, the transition graph is partitioned into separate graphs. And each subgraph is expanded as new transitions are gathered and by following the same procedure

these sub-graphs are partitioned further. So after each cut, a state is picked to be a subgoal. The disadvantages of the method are that the burden of calculations of the Max-Flow algorithm will be significant when the environment state space becomes very large and the threshold values set in advance are dependent on the problem. This parameter should be adjusted accordingly. As previously mentioned, the rooms in grid-worlds problems form strongly connected subgraphs in the transition graph and the nodes that connect these subgraphs are at focus. Strongly connected component analysis on the graph reveals all the strongly connected subgraphs (rooms) and can partition the transition graph as done in [42]. The nodes connecting the components are the ones that we are interested in. The study [42] applies strongly connected component analysis with a linear time algorithm. The algorithm requires a threshold value for which a technique is proposed in order to add an edge to the graph. L-cut [6] study approaches the subgoal identification problem from a different aspect. It aims to minimize a metric that accentuates the bottleneck nodes that connect two subgraphs. Due to the calculation burden of the metric proposed with conventional graph methods, the spectral clustering technique is employed to obtain an approximate cut. Similar to previous methods, a threshold value needs to be determined to assess the quality of the cut. If passes the test, the node (state) is picked to be a subgoal. The disadvantage of the method is that its threshold should be known in advance for the problem. Besides graph cutting algorithms, centrality metrics on the graph can be leveraged as well for the subgoal identification. The centrality concept on graphs fits pretty well for the grid-world problems since the gate regions are expected to be the most important nodes in the graph. The study [34] measures the betweenness centrality score of the nodes and picks the nodes that attain the local maximum score in their neighborhood as subgoals. Its time demand renders the algorithm unsuitable for environments with large state space sizes.

One of the main decisions that should be made with the graph-based methods is that the "sufficient information" amount (how many episodes should be stored) to identify the goal state is not known. Authors pick a certain number of episodes and run their algorithm on them. In the cases where the graph-based algorithm lacks sufficient information, it may not yield the expected results (e.g. metrics calculation failure due to insufficient edge weights, distance calculation failures due to the incomplete graph,

etc.). In the literature, methods that don't leverage graph theoretical approaches have been proposed as well. A group of them, instead, leverage the count based metrics and methods. FD algorithm [9] focuses on the states that have a higher frequency of occurring in episodes and that are distant from both the beginning and ending states of the episode. The main consideration is that the subgoal state should be present in most of the episodes (the agent must pass over them frequently) and they should be far from the starting state and ending state. The gateway locations conform to these requirements. For each state, the multiplication of the frequency and distance (by making use of a Gaussian function) values are obtained. The state that attains the highest value is considered as a subgoal. The distance metric is calculated via the trajectories of episodes by considering state appearance order in the trajectory (two consecutive state's distance is considered to be one). The algorithm possesses two parameters for measuring the distance that should be determined in advance. But the main disadvantage of the method is that it requires too many episodes to yield reliable results. Relative novelty [8] makes use of the instance occurrence in order to measure the novelty of the state (if the state is visited frequently its novelty declines). Instead of relying on the novelty metric itself, the method focuses on the relative novelty of states. When the agent transitions from one room to the other, it will be encountering new states in the new rooms and the novelty values of the state in the previous room are low because the agent has visited the states many times till it has transitioned to a new room. So the gateway locations reside between high novelty states and low novelty states. The algorithms calculate a score for each state by keeping the track of the change on the relative novelty measure depending on the novelty values of succeeding and preceding states. If the score exceeds a threshold value then the state is considered to be a subgoal. As with all methods that require predefined thresholds, this method is fragile since coming up with a single value working for all problems is a challenging task.

Subgoal identification with Diverse Density study [11] considers a totally different approach for the subgoal identification problem. It considers it as a multi-instance learning problem and leverages a well-known multi-instance learning algorithm. In the following sections, we present the study in an elaborate manner since it constitutes a base-study from which this thesis is originated. We first introduce the Diverse

Density algorithm and its necessary mathematical derivations are discussed. Later, we present subgoal identification with Diverse Density study (SDD) [5]. We emphasize important parts of the work related to our algorithm. Finally, we inspect the Expectation-Maximization Diverse Density algorithm and explain how the EMDD-RL algorithm has been derived from it.

3.3 Diverse Density

One of the earliest algorithms of the multi-instance learning paradigm is the Diverse Density DD [11, 43]. It assumes the standard assumption of MIL (negative bags should not contain any positive instance, positive bags should contain at least one positive instance) and works at the instance level. It attempts to identify the most important instance (critical instance) that best expresses the separation of the bags from each other. Its main idea stems from the fact that we might consider the bags as a path over multidimensional space and bags pass through the instances that it contains. We do expect that from a critical instance (concept instance) many positive bags should pass whereas no negative bags pass over it. This important property of an instance is measured via a probabilistic model Equation 3.1 and the instances that reside in many positive bags and do not belong to negative bags attain higher probability score (Diverse Density value). For each instance from positive bags, a probability value (DD value) is maximized from a given initial instance by considering both positive and negative bags since the instances of positive bags lead the discrimination. Further, the instance that attains the highest probability value is employed to classify the unseen bags depending on the distance between the extracted concept instance and individual instances of the unseen bag. Mathematically the maximization of the DD value from a given initial instance t can be represented by:

$$DD(t) = \max_{y} P(y = t | B_1^+ B_2^+ B_3^+ ... B_1^- B_2^- B_3^- ...) \tag{3.1}$$

where each B_i^+ and B_j^- symbols represent the i^{th} positive and j^{th} negative bags respectively. The algorithm begins from an initial instance t and performs the maximization by gradient ascent technique. The maximization procedure might end up at a location that is distant from where it has started in instance space. The point where

the algorithm has stopped is considered as an intermediate instance and this instance is considered, not the initial instance, during testing of the unseen bags.

The Equation 3.1 does not reveal the inner workings and the assumptions of the algorithm. Further, it can be expanded by performing Bayesian formula expansions by making the following assumptions. First, the instances have equal prior probabilities. Second, the bags are conditionally independent of each other for a given instance, t. Since we are comparing the score values, we can discard the normalization factor as well. Hence Equation 3.1 turns in to the following more elegant form:

$$DD(t) = \max_{y} \prod_{i}^{n^+} P(y = t|B_i^+) \prod_{j}^{n^-} P(y = t|B_j^-) \tag{3.2}$$

where n^+ and n^- stand for the number of positive and negative bags respectively. With this new formulation, it is apparent that the computations can be separated for different bag types. For representing $P(y = t|B_i^+)$ and $P(y = t|B_j^-)$ (the probability of an instance for a given bag) models authors proposed alternative ways and in [11] noisy-or model is employed and the following probability equations are obtained:

$$\begin{aligned} P(y = t|B_i^+) &= 1 - \prod_{k}^{n_i^+} (1 - P(y = t|B_{ik}^+)) \\ P(y = t|B_j^-) &= \prod_{l}^{n_j^-} (1 - P(y = t|B_{jl}^-)) \end{aligned} \tag{3.3}$$

The values are calculated by iterating through the each instance in both positive and negative bags. B_{ik}^+ represents k^{th} instance of i^{th} positive bag. Similarly B_{jl}^- is the l^{th} instance of j^{th} negative bag. n_i^+ and n_j^- are the bag sizes of i^{th} positive bag and j^{th} negative bag respectively. In Equation 3.3, we observe that in order to maximize the Equation 3.2, the tested instance should be rendered more probable by positive bag instances and less probable by the negative bag instances. In Equation 3.3 the only unrevealed necessary distribution models are the $P(y = t|B_{ik}^+)$ and $P(y = t|B_{jl}^-)$. They both model the instance to instance probabilities and are represented via the following Gaussian-like distribution:

$$P(y = t|B_{mn}) = \exp(-sd\|y - B_{mn}\|^2) \tag{3.4}$$

where B_{mn} stands for both positive and negative bags instances(n^{th} instance of m^{th} bag). Data points of the data sets are considered to be multidimensional vectors. Since

distance metrics may not reveal the true distance between instances some special care must be taken. Besides, some features on the data vector might be redundant and need to be removed. DD algorithm tackles these two problems via introducing a feature scale vector sd in its instance to instance distance calculations. This vector governs how much each feature contributes to the total distance value. This vector is considered to be a learnable parameter from the datasets since depending on the data types, the optimal feature contributions are highly likely to be different. During the maximization of DD with the gradient ascent, this parameter value is also adjusted in a such way that the DD value rises further. So the maximization of the DD value is done over both instance space and the feature scale space.

For the DD algorithm, the positive bag instances are responsible for the bag labels since it assumes the standard MIL assumption. It tests all these instances during the DD maximization. The instance that attains the highest DD value is considered as a separator for the unknown bags. Formally:

$$\begin{aligned} t^* &= \arg\max_{x \in C} DD(x) \\ C &= \{B_{ij}^+ | 1 \leq i \leq n^+, 1 \leq j \leq n_i^+\} \end{aligned} \tag{3.5}$$

So far we have considered only a single concept instance for labeling the bags. DD algorithm further supports identifying more than one concept instance as well [11].

3.3.1 Subgoal Identification with Diverse Density

Assumptions of the multi-instance learning paradigm align with the requirements of the subgoal identification in that the subgoal should be on the path to the goal and should be critical for attaining the goal. As mentioned previously, the DD's consideration of the bags as paths fits into this requirement pretty well. If we consider the episodes as bags, the diversely dense points are the ones that we are looking for since they are the critical concepts and cause for the label of the episode. Once we review the grid-world problems, obviously these critical concepts correspond to the gateway locations since the state around the gateways will reside in many positive bags and for the agent to attain the goal, they should be passed over. The Diverse Density (DD) algorithm was employed in one of the early works of subgoal identification for identifying the subgoal locations in two grid-world problems (two-room, four-room) [5].

Since the DD was developed for multi-dimensional data and the calculation burden is significant, it needed to be adapted to the discrete nature of RL tasks. Authors remove the gradient ascent step of the algorithm, instead, they check the instances' DD values (without maximization operator in Equation 3.2, sd value adjustment is removed as well). Special treatment should be made for the instance-instance distance since the data is not in multi-dimensional space and feature scaling factor sd is not learned. The authors solve this problem by calculating the graphical distance between instances on the transition graph of the environment.

Each episode is considered as a bag and the states that have been traversed are regarded as the instance of that episode. For the label of the episodes as a negative bag or positive bag, success criteria should be decided in advance (e.g. episodes ending with the goal state can be labeled as positive, the others are negative bags.) The authors leverage only positive bags for the study and no negative bags are created. Due to insufficient data amount, in the early episodes, the identified subgoals via DD algorithm might be noisy and may not be at gateway locations. Authors mitigate this problem by introducing an averaging mechanism. The instance is considered as a subgoal after getting identified sufficiently many times. As the data amount increases, stably identified concepts will start to emerge. In other words, the DD starts to find reliable subgoal locations. For identified subgoals, the options framework is used. From past trajectories of the agent, the option policies are learned. And for obtaining policies at the abstract level, Macro-Q learning [44] is employed.

3.4 Expectation-Maximization Diverse Density

Expectation-Maximization Diverse Density (EMDD) [1] algorithm approaches the DD's maximization technique from a different perspective. It treats the maximum DD valued concept as an unknown entity and formulates the problem as an Expectation-Maximization procedure. It begins the maximization procedure with initial guess for the maximum DD valued concept instance (Figure 3.1 h initialization step in $EM\text{-}DD(I, D_t)$). Throughout the EM steps, it improves its guess gradually. During its updates, the EMDD does not make use of all instances at once. It picks a single instance from every positive and negative bag that is most responsible for their bag

label by considering the current candidate concept in the expectation step of the algorithm (Figure 3.1, E step). For measuring the probability between the instances, the probability distribution in Equation 3.4) is employed. Later these selected instances are leveraged to improve the current candidate concept in such a way that label probabilities of the bag for the selected instances increases by employing DD's gradient ascent (Figure 3.1, M step). In other words, the DD value is maximized since as the DD value improves, so does the bag labeling accuracy. Similar to the DD algorithm, during maximization, sd parameter is estimated. Once the guess concept is updated, these two steps are repeated until no further DD value improvement is observed (Figure 3.1, while loop in $EM\text{-}DD(I, D_t)$).

Main(k, D)
partition $D = \{D_1, D_2, \ldots, D_{10}\}$; *//10-fold cross validation*
for $(i = 1; i \leq 10; i{+}{+})$
$D_t = D - D_i$; *//D_t training data, D_i validation data*
pick k random positive bags $B_1, \ldots, B_k$ from D_t;
let H_0 be the union of all instances from selected bags;
for every instance $I_j \in H_0$
$h_j = \textbf{EM-DD}(I_j, D_t)$;
$e_i = \min_{0 \leq j \leq \|H_0\|}\{error(h_j, D_i)\}$;
return $avg(e_1, e_2, \ldots, e_{10})$;

EM-DD(I, D_t)
Let $h = \{h_1, \ldots, h_n, s_1, \ldots, s_n\}$; *//initial hypothesis*
For each dimension $d = 1, \ldots, n$
$h_d = I_d$; $s_d = 0.1$;
$nldd_0 = +\infty$; $nldd_1 = \textbf{NLDD}(h, D_t)$;
while $(nldd_1 < nldd_0)$
for each bag $B_i \in D_t$ *//E-step*
$p_i^* = \arg\max_{B_{ij} \in B_i} Pr(B_{ij} \in h)$;
$h' = arg\max_{h \in H} \prod_i Pr(\ell_i \mid h, p_i^*)$; *//M-step*
$nldd_0 = nldd_1$; $nldd_1 = \textbf{NLDD}(h', D_t)$; $h = h'$;
return h;

Pseudo-code for EM-DD where k indicates the number of different starting bags used, $\Pr(B_{ij} \in h) = \exp[-\sum_{d=1}^{n}(s_d(B_{ijd} - h_d))^2]$. $\Pr(\ell_i \mid h, p_i^*)$ is calculate as either $1 - |\ell_i - \Pr(p_i^* \in h)|$ (linear model) or $\exp[-(\ell_i - \Pr(p_i^* \in h))^2]$ (Gaussian-like model), where $\Pr(p_i^* \in h) = \max_{B_{ij} \in B_i} \Pr(B_{ij} \in h)$.

Figure 3.1: EMDD algorithm pseudo code [1]

For identifying the concept that is most responsible for the bag labels, the DD algorithm tests all the positive bag instances as mentioned in Equation 3.5. Besides, the gradient ascent step requires a significant amount of computation as the positive instances count (as well as the total instance count) increases. The EMDD algorithm

on the other hand carries out its maximization via a lot fewer positive instances by randomly selecting instances from a few positive bags (Figure 3.1, random positive bag selection in $Main(k, D)$). Then those instances are picked to be the initial guess for the best explanatory concept. Incorporation of a lot fewer instances during gradient ascent and testing very few instances resulted in a huge speed improvement on the behalf of the EMDD algorithm compared to DD in the MIL domain (10 to 100 times faster on the tested MIL problems). Another important improvement over the DD algorithm in MIL besides the speed is that EMDD surpasses in accuracy performance. It identifies the concept state both more accurately and faster. These strong features of the EMDD algorithm have garnered our attention to study this algorithm and led to the emergence of our EMDD-RL method.

CHAPTER 4

EMDD-RL ALGORITHM

4.1 Derivation of EMDD-RL algorithm

In the multi-instance learning domain, the EMDD algorithm was shown to be superior in both accuracy and speed performances. The main cause of the speed for the algorithm is that it tests fewer instances for the peak concept search and for the DD value maximization step. During the development of the EMDD-RL, we adhere to the original algorithm strongly. But the algorithm has required some critical adjustments in order to adapt it for the RL problems. Similar to the DD algorithm, Expectation-Maximization Diverse Density was devised to work with multidimensional data and it incorporates the gradient ascent method as well. As done in SDD [5], we discard the gradient ascent step and solely compare individual instance scores for the DD maximization part (Figure 3.1, M step). The distance values between instances (states) are obtained from the transition graph. In the M-step of the algorithm (Figure 3.1) we transform the multiplication of the probability values into summation via logarithm function in order to obtain more numerically stable results. As the EMDD algorithm does, EMDD-RL (Algorithms 1-3) starts with an initial seed set selection step. The original algorithm picks randomly several positive bags and uses their instances. We instead consider the smallest positive bag (line 3) since it has to contain one of the critical concepts (subgoal) that we are looking for. By doing so, we eliminate final result oscillations due to randomized bag selection. One of the most flexible sides of the EMDD is that it allows us to consider very few instances compared to the DD algorithm. Another important fact is that the DD values of the closer states become quite close in value. These two facts have enabled us to further shrink the select the seed set. We implement this reduction by skipping some instances in the set (lines

6, 7). The fewer number of instances are tested, the faster execution time we obtain. Then we run EM steps on each selected instance (lines 11 to 17). The one that attains the highest DD value is returned as a subgoal (line 19).

In the EM part of the algorithm, the given instance is assumed to be the current most critical concept and throughout the iterations, this concept is replaced with another instance with higher DD values, finally, as no further improvement occurs in DD values, the instance that is responsible for the final improvement is returned as done in Figure 3.1. DD value maximization is carried out by considering the minimization of the log value of the DD. In the loop (line 33), the initial seed value is altered so long as the newly updated instance's DD value is higher than the previous instance.

From each bag, only a single instance that is the closest to the current guess (the concept) is identified by depending on a Gaussian-like distribution and keeping their label information in the expectation step of the algorithm (line 35-43). These selected instances constitute our $testset$ since the next guess will be picked among these instances in the maximization step of the algorithm. In the original algorithm this maximization is carried out via gradient ascent in the original instance space (Figure 3.1, M step). Instead, we abandon it, similar to SDD, and calculate the individual positive instance values since the instance with the maximum DD value should appear from the positive instances (line 46). So every $testset$ instance is tested by measuring their contribution to label probabilities individually (lines 47-67). Authors proposed different distribution models to represent the bag label probabilities for given two instances (Figure 3.1, Pseudo-code explanation part): exponential model, linear model (lines 50, 52, 57, 59). We have incorporated these two models as a hyper-parameter of the EMDD-RL (Their differences are discussed elaborately in the experiment and result parts). In addition to dropping the gradient ascent, we transformed multiplication of the probabilities into the addition (lines 51, 53, 58, and 60) by applying the logarithm function. Besides its benefits for numerical stability, taking the logarithm has enabled us to drop some exponentiation calculation when the exponential model is considered (lines 53, 60), which has considerably simplified the calculations. One significant side effect of avoiding the gradient ascent is that the newly identified instance may not have a higher DD value (previous might have a higher value but inferior instance may get returned), returning this inferior instance would be a serious mistake for the

algorithm. To sort this problem out we keep track of the instance (lines 70-73) that attains the highest score during the EMDD-RL loop and returning this instance (line 75). (Especially this concern is critical for the last iteration of the loop.)

In the original algorithm, $testset$ instances are picked from every bag by making a simple affinity check with the current guess. By getting inspired from the SDD, we have also considered all positive instances for the $testset$ instances. With this setting, all positive instances are tested for label probability maximization. The main reason for us to consider all positive instances is that the global peak might get detected before looping many times hence the algorithm execution time might get shortened. For the algorithm, we have named these two different selection strategies for $testset$ as S1 (the instances selected from the positive bag depending on their closeness to the current guess) and S2 (all instances of all positive bags). We consider them as a hyperparameter for the EMDD-RL algorithm.

For both DD and EMDD algorithms, the distance value between two instances is required in order to measure the probability of an instance (Diverse Density Equation 3.4, Figure 3.1. As a source of distance, we are making use of the agent's transition graph constructed during interacting with the environment where any two consequent states are bound with an edge carrying a distance value of 1. We consider the shortest path distance (line 22) between two states by applying Dijkstra's algorithm [45]. We avoid using the distance metrics such as Euclidean, Manhattan since the values produced by these metrics might be misleading because the obstacles residing between the start state and the end state might elongate the actual distance. Another important component of the two algorithms is the feature scaling factor, sd. Since the states are 1-d elements we do not need to seek an optimal value for this parameter (it will be affecting all the instances equally) and for both SDD algorithm and EMDD-RL algorithm calculations we set its value to 1.0 (line 23) throughout this thesis.

4.1.1 EMDD-RL Algorithm

DD value calculations are carried out via DD(t) that computes the DD value of instance t without max operator as done in SDD (Diverse Density Equation 3.2). The positive bags and negative bags are passed to this function as well but these param-

Algorithm 1 Seed Initialization and Identification of the Concept Instance

1: **function** EMDDRL(positive bags B^+, negative bags B^-, transition graph G)
2: // Construct initial seed set
3: find the smallest positive bag, $b^+_{smallest}$
4: size of the $b^+_{smallest}$ is l
5: // 0, k, 2k, 3k ... l
6: $indices = \{i \mid 0 \leq i < l, i = i + k\}$
7: $seeds = \{b^+_{smallest}[i] \mid i \in indices\}$
8: $maxddvalue = -\infty$
9: $t^* = null$
10: // Find the state that has the highest DD value
11: **for** $seed \in seeds$ **do**
12: $t, ddvalue = \text{EM}(seed, B^+, B^-, G)$
13: **if** $ddvalue > maxddvalue$ **then**
14: $maxddvalue = ddvalue$
15: $t^* = t$
16: **end if**
17: **end for**
18: // Return the state with the maximum DD value as answer
19: **return** t^*
20: **end function**

Algorithm 2 Proximity of Two Instances

21: **function** PR(instance i, instance j, transition graph G)
22: Calculate distance d between i and j on G
23: $sd = 1.0$
24: **return** $\exp(-sd.d^2)$
25: **end function**

eters are not specified in the EMDD-RL pseudo-code for convenience. k value (line 6) specifies the number of instances that will be skipped in a step-wise manner. For instance, when k is 2, the elements whose index are 0, 2, 4 ... will get picked from the smallest sized positive bag. Similarly when k is 3 the elements at 0th, 3rd, 6th ... indices are considered. So this value governs the initial seed set size. As the k value

increases, the number of elements in the seed set decreases. This value should be adjusted in such a way that neither important subgoal regions are missed nor too many instances are tested. The initial seed set should contain at least an instance closer to a region DD values of which are high so that some instances from this important region can get identified. Otherwise, the EMDD-RL algorithm may not be able to pinpoint the target regions because the Expectation-Maximization procedure may get stuck in local optima [46] and not obtain the global optimum.

Our EMDD-RL algorithm has several hyperparameters to be decided: k value (skipping factor), $testset$ instance selection strategy, and label probability model (exponential, linear). The first hyperparameter is the skipping factor, k (line 6). When k is a big number, more instances will be eliminated from the initial seed set. DD values exhibit locality (DD values of neighbor states are close to each other) and this enables us to eliminate several states over the neighborhood, if elimination is done severely, we might miss a region where the global DD value peak occurs during maximization. Since the EMDD algorithm may get stuck in local peak points, we need to tweak this parameter not to skip global peaks. The second hyperparameter is the selection of the distribution of the label probabilities: linear or exponential model (lines 50, 52, 57, 59). Third and the final hyperparameter is the set of instances ($testset$ variable in the algorithm, line 46) that are checked in the M step. In the original EMDD algorithm, maximum is found over the selected instances by applying gradient ascent whereas we evaluate each selected positive instance (the instances that are picked from each bag by measuring their probability with the current guess) and pick the one that attains the highest value of summation of the log probabilities. Besides, we have proposed a different way of forming the $testset$: all positive instances. Our intuition behind this set is that including all positive instances might lead to faster global peak detection. Peak instance might get selected earlier and loop execution might last shorter hence a faster execution time might be obtained.

4.1.2 Bag Preparation and Transition Graph Construction

Adhering to SDD [5], we prepare positive and negative bags from the episode depending on whether they have ended in the goal state (positive bags) or not (negative

bags). The agent can keep track of the states it has traversed in every episode and at the end of each episode, the bag of the episode can be prepared and stored. The states that are traversed in a particular episode are regarded as the instances of the bag of the episode. Since we are working in 1-d states (throughout this thesis) some states will be redundant in episodes (e.g. some of the states can be visited more than one time and appear in the episode trajectory multiple-times.) For both the SDD algorithm and EMDD-RL algorithm, only a single value calculation for a particular state is sufficient. So we remove redundant instances and keep only the distinct states for both negative and positive bags. EMDD-RL algorithm requires a transition graph to measure distances between instances. This graph is constructed by inspecting the trajectory history of the agent for all episodes. If two states appear to be adjacent to each other in the trajectory, these two states are added to the transition graph if they are not present yet and an edge is created between them with a weight of 1 indicating the distance value between states.

Algorithm 3 Expectation-Maximization

function EM(instance seed, positive bags B^+, negative bags B^-, transition graph G)
  $nldd0 = \infty$
  $nldd1 = -\log(\mathrm{DD}(seed))$
  $peakstate = seed$
  $highestDD = -1 * nldd1$
  // Keep updating current guess as long as DD value improves
  //Main Loop
  while $nldd1 < nldd0$ **do**
    // E Step
    $p^*_+ = set(), p^*_- = set()$
    for each bag $B^+_i \in B^+$ **do**
      $p = \arg\max_{B_{ij} \in B_i} \mathrm{PR}(B_{ij}, peakstate, G)$
      append p to p^*_+
    end for
    for each bag $B^-_k \in B^-$ **do**
      $p = \arg\max_{B_{kl} \in B_k} \mathrm{PR}(B_{kl}, peakstate, G)$
      append p to p^*_-
    end for
    // M Step
    $h' = null$; $maxhvalue = -\infty$
    $testset = \{p_i | p_i \in p^*_+\}$
    for each instance $t_i \in testset$ **do**
      sum = 0.0
      for $p_i \in p^*_+$ **do**
        if Linear model is picked **then**
          $sum+ = \log(1 - |1.0 - \mathrm{PR}(p_i, t_i, G)|)$
        else if Exponential model is picked **then**
          $sum+ = -(1.0 - \mathrm{PR}(p_i, t_i, G))$
        end if
      end for
      for $p_l \in p^*_-$ **do**
        if Linear model is picked **then**
          $sum+ = \log(1 - \mathrm{PR}(p_l, t_i, G))$
        else if Exponential model is picked **then**
          $sum+ = \mathrm{PR}(p_l, t_i, G)$
        end if
      end for
      if $sum > maxhvalue$ **then**
        $sum = maxhvalue$
        $h' = t_i$
      end if
    end for
    $nldd0 = nldd1$
    $nldd1 = -\log(\mathrm{DD}(h'))$
    if $nldd1 < nldd0$ **then**
      $peakstate = h'$
      $highestDD = -1 * nldd1$
    end if
  end while
  return $peakstate$, $highestDD$
end function

CHAPTER 5

EXPERIMENTS AND RESULTS

5.1 Experiments

In order to obtain a better understanding of how superior the EMDD-RL is over SDD and how the hyperparameters of the EMDD-RL affect the performance of the algorithm on the subgoal identification problem, we have conducted three different experiments: speed experiment, state-space effect experiment, accuracy experiment.

For speed experiments, we have gathered state trajectories from two-room and four-room environments with two different instance collection strategies in order to be able to create positive bags and negative bags. In the first one, we have not restricted the agent with a step limit for an episode and in the second one, 200 and 400 step limits are imposed for two-room and four-room respectively. In a step limited episode, if the agent cannot reach the goal state by exceeding the step limit (200 steps for two-room, 400 steps for four-room), the episode is finalized and the agent starts a new episode. Step limitless and limited trajectories are gathered from 100 trials separately, where each trial consists of at least 100 episodes. For gathering all data from both environments we have made use of a Q-learning agent with the same parameters as SDD [5], epsilon $(\epsilon) = 0.1$, learning rate $(\alpha) = 0.05$, discount factor $(\gamma) = 0.9$ and the agent is given 4 different actions to apply (left, right, up, down) with action noise (moving in intended direction with 90% probability and with 10% probability in any direction). The agent has received a reward of 0 in non-goal states and a reward of 1 in the goal state.

We have formed three different datasets for both trajectory collections separately. From step-limited trajectories, STEPBALANCED and STEPPOSITIVE datasets have

been prepared. STEPBALANCED dataset consists of the first 20 unsuccessful (not ending with the goal state) episodes of the trial (as negative bags) and the first 20 successful (ending with the goal state) episodes of the trial (as positive bags). STEP-POSITIVE dataset is formed with the first 20 successful episodes of the trial (as positive bags) and no negative bag is considered. In a similar way, from step limitless trajectories POSITIVE dataset has been created, which contains the first 20 episodes of the trial (as positive bags) and no negative bag is created. STEPBALANCED, STEPPOSITIVE, POSITIVE datasets are prepared for both two-room and four-room separately. EMDD-RL and SDD have been tested on these datasets gathered from two-room and four-room for speed comparisons.

In order to clarify the speed experiment conduction, assume that we make use of the two-room environment and consider step limited trajectories. We run the agent with the given configuration above in the two-room. After completing a specific number of episodes (sufficient data is gathered for constructing positive and negative bags), we record the agent's trajectories for the current trial and then repeat the same procedure 99 times more. After obtaining all data, during the speed comparisons, from every recorded trial STEPBALANCED (first 20 negative trajectories are considered to be negative bags, first 20 positive trajectories are considered to be positive bags) and STEPPOSITIVE (first 20 positive episodes are considered to be positive bags, no negative bag is created.) datasets are constructed. Then the SDD algorithm is run over STEPPOSITIVE datasets constructed for each trial one by one. After running over the 100 STEPPOSITIVE datasets, the speed results of the SDD are averaged for the 100 trials (Instances that are picked to be the goal state in each trial are used to measure the accuracy metric.). The reason why we average is to obtain general, reliable accuracy and execution speed results that are independent of the trajectories of the trials. In the same way, both algorithms are run over all datasets for both environments. Hence each algorithm obtains 100 results (for execution speed) for each dataset in each environment and the averaged results are used for comparing the algorithms.

In the state space effect experiment, we would like to inspect how the speed performances of the algorithms change as the state space of the two-room environment enlarges. We have built extra 5 two-room environments with different state sizes

(Rooms are expanded horizontally, the wall is kept in the middle of the environment) and datasets are built in the same way as the speed experiment.

The first aim of the accuracy experiment has been to compare the accuracy performances of the SDD and EMDD-RL algorithms when the data amount is reduced. Basically, the algorithm that attains better accuracy via fewer instances can be easily preferred to the other. The second aim of this experiment has been to determine how the EMDD-RL parameters affect the speed and accuracy performances of the algorithm. The third aim has been to grasp an idea of how negative bag usage contributes to algorithms performances (In the SDD study, this question was not asked). For these purposes, we have tested the EMDD-RL and SDD algorithms with varying bag sizes and parameters (The SDD algorithm does not possess a special parameter to be tweaked). We have leveraged the STEPBALANCED dataset of the speed experiment for both two-room and four-room environments. The positive bag sizes have varied as (5, 10, 15, 20) whereas the negative bag size has been taken from the set (0, 5, 10, 15, 20). For the EMDD-RL algorithm, strategies S1 and S2, skipping factor values from the set (1, 2, 3, 4), linear and exponential models have been tested. All combinations of bag sizes and algorithms' parameters have been considered.

In all experiments to assess the quality of the subgoals found by both algorithms, we calculate the accuracy metric. Simply we count how many locations we expect them to be selected in the environment as a subgoal, states that surround the gateways have been truly identified, and divide this count with the total number of predictions (Since each algorithm is run 100 times over a particular dataset, 100 subgoal prediction results have been obtained for that particular dataset. We count how many predictions have been successful among these 100 predictions). The expected locations are shown in Figure 5.1. For the four-room environment, we have also considered the states in the affinity of the goal state since these locations are highly like to be visited more frequently as well. Hence DD values of these states are high.

For all experiments, we have made use of a transition graph extracted directly from the environment not from trajectories in order to eliminate the problems that might occur due to incomplete node connections changing from trial to trial (i.e. incorrect distance values between nodes), which might affect the performances of the algorithms. So

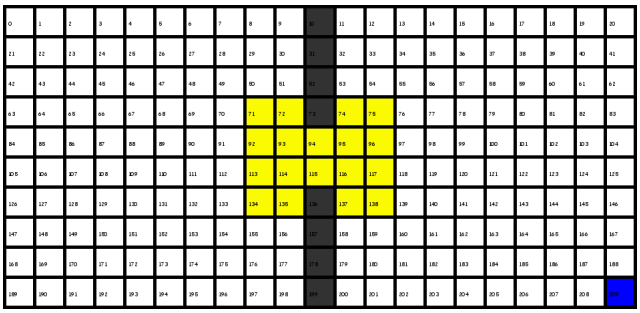
both algorithms have been compared in equal conditions.

In order to show that the EMDD-RL algorithm can be used to accelerate the problem solving in RL, we have combined it with the options framework and compared the learning performance of it with a Q-learning agent. The performance comparison has been carried over depending on the step count of the agent to reach the goal state from a particular starting state with both methods.

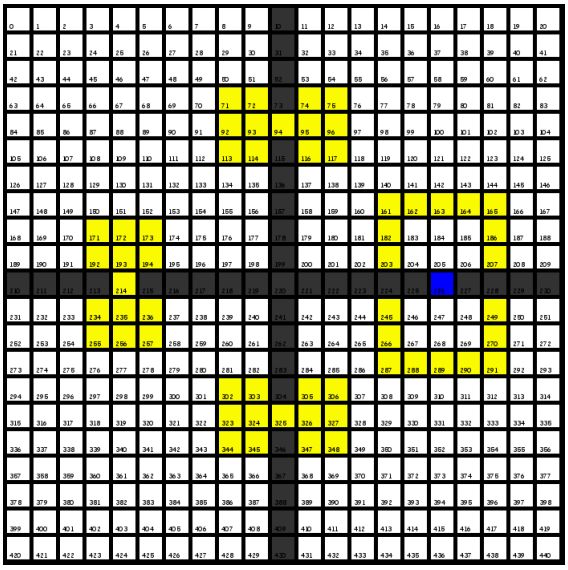
5.2 Results

Here we present the results of speed, state effect, accuracy experiments conducted over 3 datasets of each environment by using SDD and EMDD-RL algorithms. For each trial, we have created 3 datasets as described in the previous section, and on every trial, EMDD-RL and SDD algorithms have been run (total 100 times running on each dataset). For speed performance assessment, these experiments have been repeated 10 times (A single experiment run averages over 100 results on a dataset, the same experiment run is repeated 9 times more, and final results are obtained by averaging the 10 run results) for each algorithm and configuration on each dataset in order to obtain reliable run time results cleansed from execution oscillations due to operating system, and the running time results shown are averaged. The EMDD-RL algorithm has three hyperparameters to be tweaked: label probability model (exponential model, EXP; linear model, LIN), search strategy (selected positive instances, S1; all positive instances, S2), and elimination factor (k, we have tested values of 1, 2, 3, 4) for trimming seed instance set.

In Figure 5.2, log(DD) values of the states are shown on both environments. For each dataset, the DD values of the states are calculated over 100 trials and averaged. The brightness of the green color indicates the DD value. As the brightness increases, so does the DD value. The gateway locations exhibit higher DD values compared to the rest.

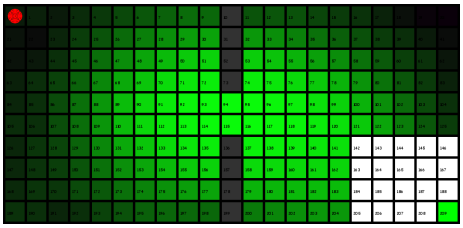


(a) Two-room Environment

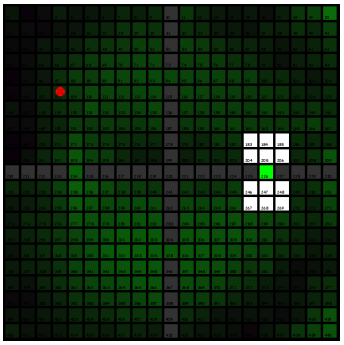


(b) Four-room Environment

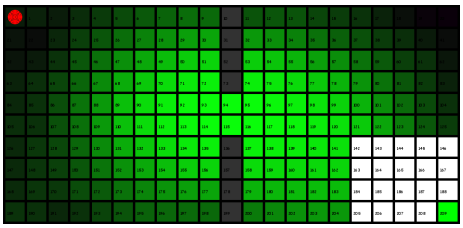
Figure 5.1: Environments with enumerated cells are shown. The yellow color indicates the expected subgoal locations for both environments. Blue-colored locations are goal states. If the subgoal state found by the algorithms matches with any of the yellow-colored locations, the result is deemed as a success. In both environments, the states around the goal state are eliminated for avoiding DD value bias toward these states. For the four-room, the states surrounding these eliminated states are also strong candidates for subgoal since positive trajectories have to pass over these states.



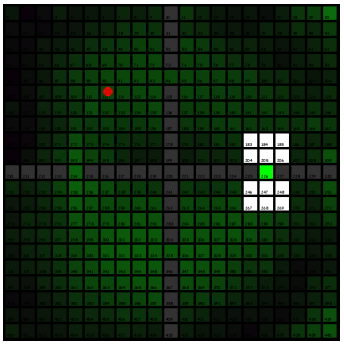
(a) STEPBALANCED



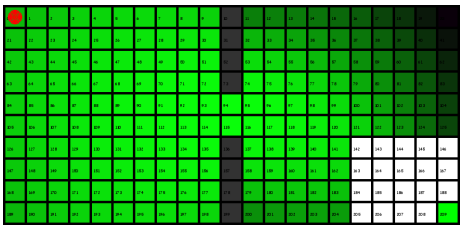
(b) STEPBALANCED



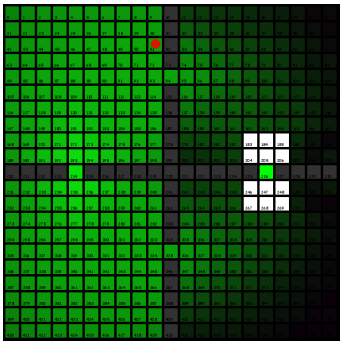
(c) STEPPOSITIVE



(d) STEPPOSITIVE



(e) POSITIVE



(f) POSITIVE

Figure 5.2: $log(\text{DD})$ value maps of all states of two-room and four-room. The DD values are calculated with the corresponding datasets over 100 trials. The values are scaled between 0-255 for displaying as a color.

5.2.1 Speed Experiment Two-room Environment

In Tables 5.1 and 5.2, we present the averaged speed results of SDD and EMDD-RL algorithms respectively. For the EMDD-RL algorithm, we show the fastest configu-

ration that attains the same or better accuracy compared to SDD and the configuration that attains the highest accuracy for all datasets of two-room.

The fastest EMDD-RL algorithm has been obtained when the exponential model (EXP) is employed and the current guess is improved by picking among selected positive instances (S1) for all datasets and all k values.

In order to be able to explain the EMDD-RL hyperparameter effect on the execution speed, we also show the average loop count in the EM step for all configurations. When the exponential model is employed, the fewest loop count for the EM step is obtained. The exponential model is superior to the linear model in speed for all datasets, both search strategies (S1, S2) and for all k values tested. The strategy that improves the current guess among selected positive instances (S1) always attains the shortest execution time against the other strategy (picking among all positive instances, S2) for all datasets, both linear and exponential models, and all k values tested. Depending on the k value algorithm accelerates significantly, it should be neither too big nor too small for fast execution time and accuracy performances. A value of 2 or 3 for k is quite reasonable for grid problems.

When we consider the fastest configurations that attain the same or better accuracy (shaded configurations on the table), EMDD-RL runs 6.30 times faster on the STEPBALANCED dataset, 5.30 times faster on the STEPPOSITIVE dataset, 3.37 times faster on the POSITIVE dataset of two-room compared to SDD algorithm. With respect to the accuracy, the EMDD-RL algorithm performs as well as SDD on STEPBALANCED and STEPPOSITIVE datasets whereas on the POSITIVE dataset EMDD-RL algorithm is able to obtain better accuracy.

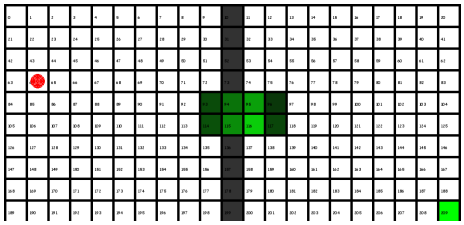
The identified subgoals for both algorithms and their configurations on two-room environment datasets are depicted in Figure 5.3. For each dataset 100 subgoals have been identified. The identification frequency of a particular state is represented via green color. The brighter the green color is, the more times that state has been picked to be a subgoal.

Table 5.1: Speed experiment results of the SDD algorithm for each dataset of two-room. The accuracy metric and average running time (in seconds) are shown.

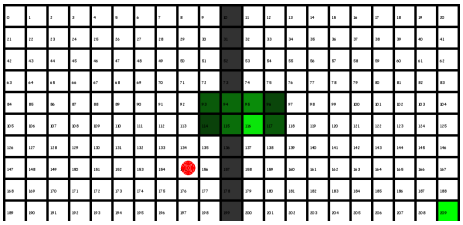
SDD performance (two-room)		
Dataset	Accuracy	Time
STEPBALANCED	1	0.6480
STEPPOSITIVE	1	0.3169
POSITIVE	0.78	0.6650

Table 5.2: Speed experiment results of the EMDD-RL algorithm for each dataset of two-room. The shaded rows are the fastest configurations that improve or perform as well as the SDD in accuracy metric. Non-shaded rows represent the EMDD-RL configuration that has attained the highest accuracy performance on the corresponding dataset. The time column depicts the average running time (in seconds) of the configuration.

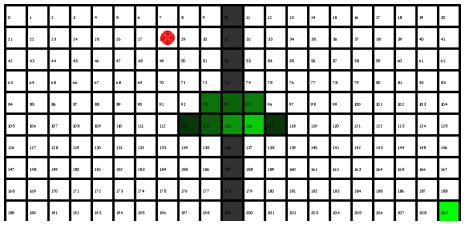
EMDD-RL performance (two-room)						
Dataset	Strategy	Model	k	Loop	Accuracy	Time
STEPBALANCED	S1	EXP	3	11.96	1.00	0.1028
STEPBALANCED	S1	EXP	3	11.96	1.00	0.1028
STEPPOSITIVE	S1	EXP	3	13.08	1.00	0.0598
STEPPOSITIVE	S1	EXP	3	13.08	1.00	0.0598
POSITIVE	S2	EXP	4	19.06	0.78	0.1970
POSITIVE	S2	EXP	3	25.09	0.79	0.2624



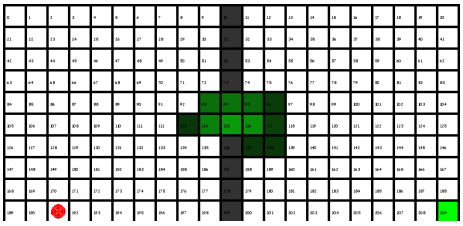
(a) SDD result on STEPBALANCED



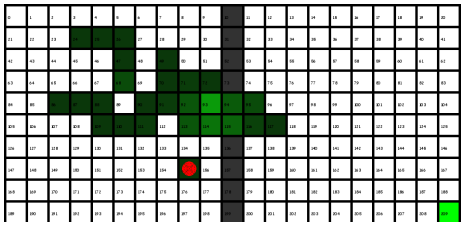
(b) EMDD-RL result on STEPBALANCED



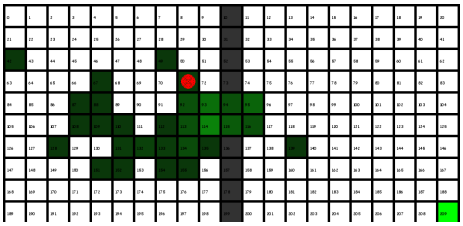
(c) SDD result on STEPPOSITIVE



(d) EMDD-RL result on STEPPOSITIVE



(e) SDD result on POSITIVE



(f) EMDD-RL result on POSITIVE

Figure 5.3: Visual outputs of the algorithms in Tables 5.1, 5.2 for the two-room environment. The EMDD-RL results are of the shaded configurations in Table 5.2. The identified states are shaded depending on their frequency out of 100 trials (Brightness of the state increases as its identification frequency increases). The red circle represents the agent.

5.2.2 Speed Experiment Four-room Environment

In Tables 5.3 and 5.4, the speed performances of the algorithms on four-room environment are shown. The same observations (speed superiority of exponential model and search strategy S1, k value effect) that we have made for the EMDD-RL algorithm on two-room are also valid for the four-room environment. Similarly, when we consider the fastest configurations that attain the better or the same accuracy (shaded configurations on the table), EMDD-RL delivers 10.85, 8.69, 4.89 times faster execution duration on STEPBALANCED, STEPPOSITIVE, POSITIVE datasets of four-

room respectively. EMDD-RL algorithm attains better accuracy results on STEP-BALANCED and STEPPOSITIVE datasets whereas on the POSITIVE dataset results of both algorithms are the same.

The identified subgoals for both algorithms and their configurations on four-room environment datasets are depicted in Figure 5.5. For each dataset 100 subgoals have been identified. The identification frequency of a particular state is represented via green color. The brighter the green color is, the more times that state has been picked to be a subgoal.

Table 5.3: Speed experiment results of the SDD algorithm for each dataset of four-room. The accuracy metric and average running time (in seconds) are shown.

SDD performance (four-room)		
Dataset	Accuracy	Time
STEPBALANCED	0.89	1.9020
STEPPOSITIVE	0.89	0.9129
POSITIVE	0.88	2.1361

5.2.3 State Effect Experiment

This experiment has been carried out to inspect how the speed performances of two algorithms get affected by the state space size of the problem. Both algorithms have been tested on two-room environments with state-space sizes of 210 (original), 310, 410, 510, 610, 710 (Figure 5.6). For the datasets that require step limitation, the step limits applied are proportional to the environment sizes and as follows: 200, 300, 400, 500, 600, 700. In Figure 5.4, the speed performances of the algorithms are shown. For speed comparisons on the datasets, configurations of the EMDD-RL algorithm that attain better or the same accuracy performance as the SDD algorithm are considered. For all datasets, as the state space of the environment expands, the speed performance gap between algorithms enlarges. The space growth effect on EMDD-RL is significantly smaller compared to the SDD algorithm, which renders

Table 5.4: Speed experiment results of the EMDD-RL algorithm for each dataset of four-room. The shaded rows are the fastest configurations that improve or perform as well as SDD in accuracy performance. Non-shaded rows represent the configuration that has attained the highest accuracy performance on the corresponding dataset. The time column depicts the average running time (in seconds) of the configuration.

EMDD-RL Performance (four-room)						
Dataset	Strategy	Model	k	Loop	Accuracy	Time
STEPBALANCED	S1	EXP	3	14.73	0.90	0.1753
STEPBALANCED	S1	EXP	2	21.42	0.92	0.2546
STEPPOSITIVE	S1	EXP	3	46.40	0.91	0.1051
STEPPOSITIVE	S2	EXP	1	46.40	0.94	0.5580
POSITIVE	S1	EXP	3	41.89	0.88	0.4370
POSITIVE	S1	EXP	3	41.89	0.88	0.4370

the EMDD-RL algorithm a strong candidate for the environments with large state space sizes.

5.2.4 Accuracy Experiment

In this section, we present accuracy experiment results with three different aspects: superior algorithm, negative bag effect, EMDD-RL parameter effect. In the superior algorithm part, the accuracy results of SDD and EMDD-RL are compared for each positive and negative bag count. For the EMDD-RL algorithm, we consider the most accurate parameter setting for particular positive and negative bag sizes, and this setting is compared with the SDD algorithm. In the negative bag effect part, we inspect how the number of negative bags affects the performance of the algorithms separately. Finally, in the parameter effect part, we present how skipping factor, selected model and instance selection strategies contribute to the EMDD-RL's overall performance on a particular positive and negative bag size.

In the cases where a wide-spread behavior has not been observed or a general deduc-

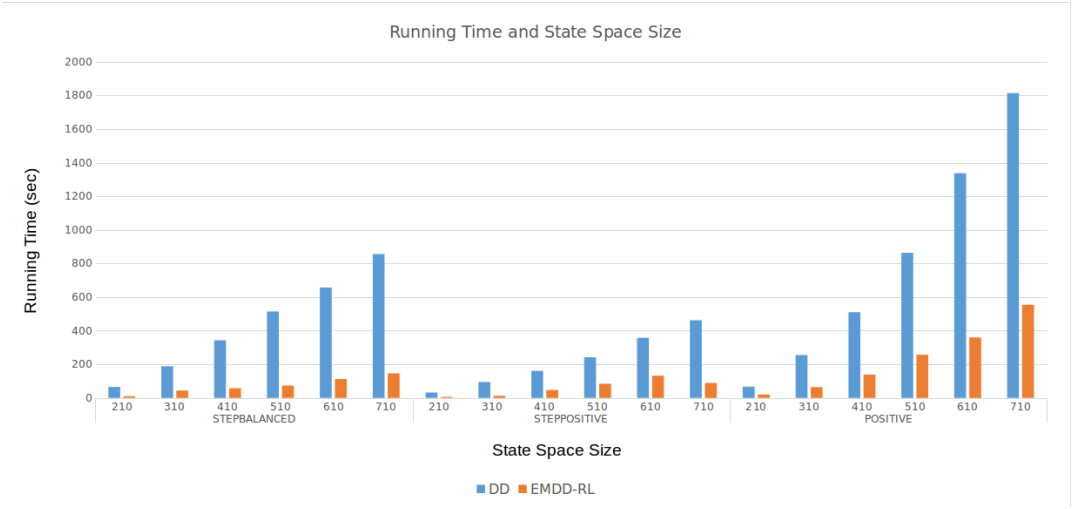


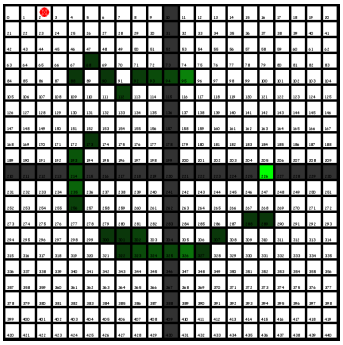
Figure 5.4: Barchart graphs of running time performance of the SDD and EMDD-RL algorithms on the two-room environment with different state space sizes. The charts are partitioned concerning the datasets. The accuracy performance of the EMDD-RL algorithm shown is better or the same as the SDD algorithm for all datasets. The Running Time axis shows the total time of the 100 runs of a particular algorithm.

tion has not been obtained, we present the conflicting cases to refute candidate claims which we expect to observe for each part. For complete accuracy experiment results, you may refer to Appendix.

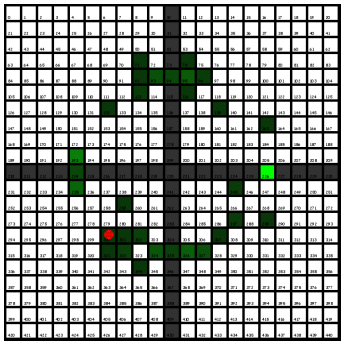
For both environments, the STEPBALANCED dataset used in the speed experiment is considered since this is the only dataset that contains both negative and positive data bags.

5.2.5 Accuracy Experiment (Superior Algorithm)

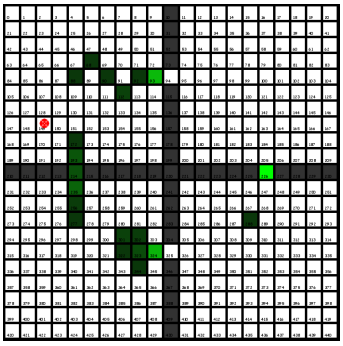
Tables 5.5 and 5.6 depict the accuracy performances of the algorithms for the two-room and four-room environments. The fastest EMDD-RL setting which attains the best accuracy score has been picked for the comparison. Depending on the EMDD-RL accuracy results, the rows are colored. If EMDD-RL performs better its row color is green. If the results of both algorithms are identical, no color is used. And for the cases it fails to surpass the SDD algorithm, the row is marked with red. In the



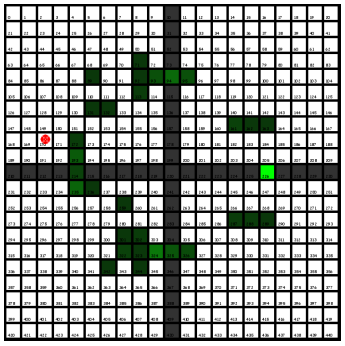
(a) SDD result on STEPBALANCED



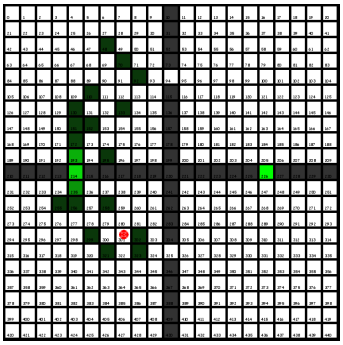
(b) EMDD-RL result on STEPBALANCED



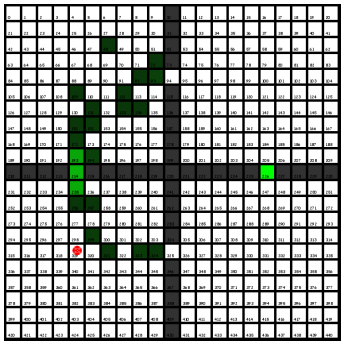
(c) SDD result on STEPPOSITIVE



(d) EMDD-RL result on STEPPOSITIVE



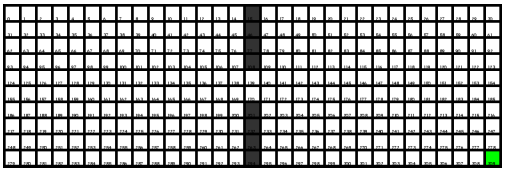
(e) SDD result on POSITIVE



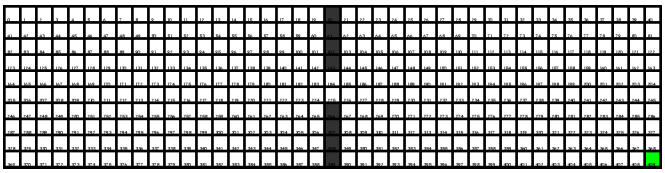
(f) EMDD-RL result on POSITIVE

Figure 5.5: Visual outputs of the algorithms in Tables 5.3, 5.4 for the four-room environment. The EMDD-RL results are of the shaded configurations in Table 5.4. The identified states are shaded depending on their frequency out of 100 trials (Brightness of the state increases as its identification frequency increases). The red circle represents the agent.

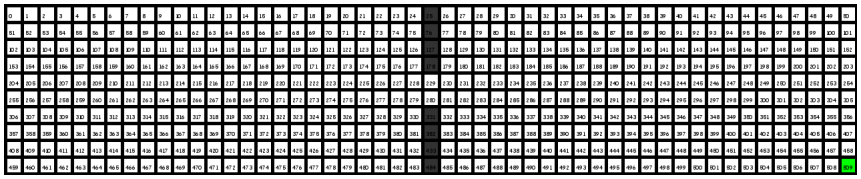
two-room environment, for most of the bag size combinations, EMDD-RL performs better in accuracy performance (For only one case, 5 positive bags and 0 negative bag,



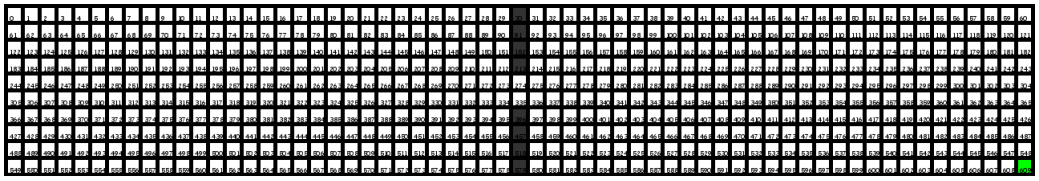
(a) Two-room environment with 310 states. 300 step limit is imposed for the datasets requiring a step limit.



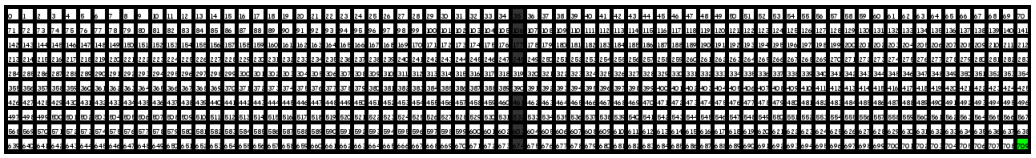
(b) Two-room environment with 410 states. 400 step limit is imposed for the datasets requiring a step limit.



(c) Two-room environment with 510 states. 500 step limit is imposed for the datasets requiring a step limit.



(d) Two-room environment with 610 states. 600 step limit is imposed for the datasets requiring a step limit.



(e) Two-room environment with 710 states. 700 step limit is imposed for the datasets requiring a step limit.

Figure 5.6: Environments that have been used for the state space effect experiment are shown with enumerated states. The environments are elongated horizontally after each increase in the space size. The green-color states indicate the goal state of the environment.

it fails). On the other hand, it outperforms in accuracy for all other cases.

In some cases (e.g. 5 positive bags and 0 negative bag) the running time performance of the SDD algorithm is better. It stems from the fact that with small instance counts it is very natural for EMDD-RL to be slow because its calculation overhead overwhelms its speed performance. In other words, checking all positive instances requires less computation time with small instance counts.

The run time columns of the tables depict the 100 runs average time for the algorithms. These values may not be identical via speed experimentation (e.g. 20 positive bags, 20 negative bags) because the speed experimentation has been repeated 10 times whereas the accuracy experiment has been run only once since the accuracy value of the algorithms does not change when the same experiment is repeated. Although these values are not precise (needs to be repeated many times to obtain more reliable results, as done in the speed experiment) still these run time values provide a strong idea about the speed performance of the algorithms.

These results suggest to us that the EMDD-RL algorithm is capable of surpassing the SDD algorithm for the subgoal identification problem in both speed and accuracy as it did for the problems of the MIL.

Table 5.5: Accuracy comparisons of EMDD-RL and SDD algorithms on two-room environment. The cases where EMDD-RL excels is marked via green color. If they have identical accuracy performance, no coloring scheme is used. The other failure cases are specified via red color. The unit of the Time column is second. For the two-room, in most of the bag size combinations, the EMDD-RL algorithm excels in accuracy performance or performs as well as the SDD. Only for one setting, it has failed to surpass the SDD algorithm.

Accuracy Performance (two-room)							
Alg	Pstv	Ngtv	Strg	Model	Skip	Accuracy	Time
SDD	5	0	S2	EXP	2	0.92	0.0724
EMDDRL						0.86	0.0740
SDD	5	5				0.70	0.1365

Table 5.5: (continued)

EMDDRL			S2	EXP	4	0.81	0.0688
SDD	5	10				0.69	0.1971
EMDDRL			S1	LIN	4	0.78	0.0854
SDD	5	15				0.60	0.2618
EMDDRL			S2	EXP	4	0.76	0.1263
SDD	5	20				0.57	0.3271
EMDDRL			S2	EXP	3	0.66	0.1658
SDD	10	0				0.98	0.1591
EMDDRL			S1	EXP	2	0.99	0.0687
SDD	10	5				0.97	0.2268
EMDDRL			S2	EXP	3	0.97	0.0984
SDD	10	10				0.96	0.3039
EMDDRL			S2	EXP	3	0.96	0.1245
SDD	10	15				0.93	0.3939
EMDDRL			S1	EXP	3	0.95	0.0849
SDD	10	20				0.93	0.4649
EMDDRL			S1	EXP	1	0.93	0.3054
SDD	15	0				1.00	0.2386
EMDDRL			S1	EXP	2	1.00	0.0792
SDD	15	5				0.99	0.3123
EMDDRL			S1	EXP	4	0.99	0.0575
SDD	15	10				0.98	0.3895
EMDDRL			S1	EXP	1	0.99	0.2194
SDD	15	15				0.98	0.4711
EMDDRL			S1	EXP	1	0.99	0.2492
SDD	15	20				0.98	0.5564
EMDDRL			S1	EXP	2	0.98	0.1553
SDD	20	0				1.00	0.3198
EMDDRL			S1	EXP	3	1.00	0.0588
SDD	20	5				1.00	0.3906

Table 5.5: (continued)

EMDDRL			S1	EXP	4	1.00	0.0639
SDD	20	10				1.00	0.4823
EMDDRL			S1	EXP	3	1.00	0.0790
SDD	20	15				1.00	0.5792
EMDDRL			S1	EXP	3	1.00	0.0913
SDD	20	20				1.00	0.6416
EMDDRL			S1	EXP	3	1.00	0.1040

Table 5.6: Accuracy comparisons of EMDD-RL and SDD algorithms on four-room environment. The cases where EMDD-RL excels in accuracy performance is marked via green color. If they have identical accuracy performance, no coloring scheme is used. The other failure cases are specified via red color. The unit of the Time column is second. For the four-room in all of the bag size combinations, the EMDD-RL algorithm excels in accuracy performance.

Accuracy Performance (four-room)							
Alg	Pstv	Ngtv	Strg	Model	Skip	Accuracy	Time
SDD	5	0				0.57	0.1865
EMDDRL			S1	LIN	4	0.63	0.0529
SDD	5	5				0.56	0.3678
EMDDRL			S1	EXP	4	0.72	0.0836
SDD	5	10				0.53	0.5455
EMDDRL			S2	LIN	4	0.70	0.4752
SDD	5	15				0.52	0.6933
EMDDRL			S2	EXP	3	0.65	0.2796
SDD	5	20				0.55	0.8511
EMDDRL			S2	EXP	3	0.66	0.3285
SDD	10	0				0.83	0.4402

Table 5.6: (continued)

EMDDRL			S2	EXP	3	0.86	0.1626
SDD	10	5				0.83	0.6784
EMDDRL			S1	EXP	3	0.87	0.1124
SDD	10	10				0.80	0.9291
EMDDRL			S1	EXP	4	0.84	0.1215
SDD	10	15				0.84	1.1165
EMDDRL			S2	EXP	4	0.87	0.2976
SDD	10	20				0.85	1.3538
EMDDRL			S2	EXP	4	0.87	0.3416
SDD	15	0				0.85	0.7088
EMDDRL			S1	LIN	4	0.87	0.0971
SDD	15	5				0.85	0.9252
EMDDRL			S1	EXP	2	0.90	0.1683
SDD	15	10				0.85	1.2187
EMDDRL			S1	EXP	3	0.91	0.1404
SDD	15	15				0.84	1.4033
EMDDRL			S1	EXP	2	0.91	0.2291
SDD	15	20				0.82	1.6698
EMDDRL			S2	EXP	1	0.91	1.1150
SDD	20	0				0.89	0.9411
EMDDRL			S2	EXP	1	0.94	0.5626
SDD	20	5				0.90	1.1590
EMDDRL			S1	EXP	1	0.92	0.3197
SDD	20	10				0.90	1.4012
EMDDRL			S1	EXP	1	0.94	0.3754
SDD	20	15				0.91	1.6837
EMDDRL			S2	EXP	3	0.93	0.3392
SDD	20	20				0.89	1.9210
EMDDRL			S1	EXP	2	0.92	0.2575

5.2.6 Accuracy Experiment (EMDD-RL Parameter Effect)

In speed experiments (20 positive bags, 20 negative bags have been considered.) the EMDD-RL setting with exponential model and S1 selection strategy has attained better or similar accuracy values and faster execution times compared to SDD algorithm in two-room and four-room environments. And in the previous section results, the EMDD-RL algorithm surpasses the SDD algorithm in accuracy performance on most of the data settings for the two-room and four-room environments. As can be seen in Tables 5.5 and 5.6, the fastest and the most accurate parameter setting changes depending on the data and there is no dominant setting for all data configurations. When we consider the only speed criteria the exponential model always attains the faster execution times compared to the linear model except for very small data bags (5 positive bags and 0 negative bag setting in two-room, 5 positive bags and 0 negative bag, and 5 positive bags and 5 negative bags settings in four-room), strategy S1 is always faster than strategy S2. But when we take the accuracy into account as we have seen in the previous section, different k values, different strategies (S1, S2), different model settings (exponential, linear) attain different results. This stipulates us to find out the best EMDD-RL setting for the problem that will be solved. In other words, depending on the data, the parameter search is suggested to pick the best performing EMDD-RL setting for the accuracy performance. By adhering to the speed experiment, it is quite reasonable to try the setting with strategy S1 and exponential model out at first attempt.

5.2.7 Accuracy Experiment (Negative Bag Effect)

In the two-room environment, for the SDD algorithm the negative bag usage deteriorates or does not change the accuracy results for a particular positive bag size and as the positive bag number increases the SDD algorithm becomes less affected by the negative bag count with the results depicted in Table 5.5. For the four-room environment, the effect is not consistent in Table 5.6. For bag sizes 5, 10, 20 the negative bag

addition degrades the accuracy up to some extent but further adding might result in better accuracy. For a positive bag size of fifteen, we observe the same behavior as in the two-room environment.

For the EMDD-RL algorithm, here we consider the setting with exponential model and strategy S1 since from the speed experiments we know that this attains the fastest speed performance. In the two-room environment, for the k (skipping factor) values of 1, 3, and 4 we can easily observe in Table 5.7 that there is no consistent effect on accuracy performances, for some positive bag counts further negative bag inclusion leads accuracy degradation whereas, for some other, the accuracy performance might improve or remain the same. For instance when $k = 1$, for the positive bag size of 10 as the negative bag size increases the accuracy performance declines. In the contrast, for the positive bag size of 15, the accuracy fluctuates. Similar facts for k values of 3 and 4 can be observed as well. For $k = 2$, the accuracy performance decreases or remains the same. For the four-room environment, the accuracy of the EMDD-RL setting fluctuates with negative bag inclusion in Table 5.8. The accuracy may increase or decrease for each k value.

For both SDD and EMDD-RL algorithms, in both environments, a consistent effect cannot be observed in both positive or negative manners. When we consider the overall results, the main accuracy source seems to be the positive bag count. As the positive bag count increases the effect of the negative bag diminishes for both algorithms. The addition of negative bags might improve or degrade accuracy performance depending on the data and the hyperparameters.

Table 5.7: Negative bag effect on EMDD-RL for the two-room environment. For different k values of the setting with strategy S1 and exponential model, the accuracy performances are depicted.

EMDD-RL (S1, EXP) Negative Bag Effect (two-room)						
Pstv	Ngtv	Strg	Model	Skip	Accuracy	Time
10	0	S1	EXP	1	0.98	0.1322
10	5	S1	EXP	1	0.97	0.1637
10	10	S1	EXP	1	0.95	0.2042

Table 5.7: (continued)

10	15	S1	EXP	1	0.94	0.2530
10	20	S1	EXP	1	0.93	0.3054
15	0	S1	EXP	1	1.00	0.1513
15	5	S1	EXP	1	0.98	0.1812
15	10	S1	EXP	1	0.99	0.2194
15	15	S1	EXP	1	0.99	0.2492
15	20	S1	EXP	1	0.97	0.2937
20	0	S1	EXP	1	1.00	0.1703
20	5	S1	EXP	1	1.00	0.1921
20	10	S1	EXP	1	1.00	0.2213
20	15	S1	EXP	1	1.00	0.2687
20	20	S1	EXP	1	1.00	0.3003
10	0	S1	EXP	2	0.99	0.0687
10	5	S1	EXP	2	0.95	0.0839
10	10	S1	EXP	2	0.95	0.1048
10	15	S1	EXP	2	0.95	0.1258
10	20	S1	EXP	2	0.92	0.1524
15	0	S1	EXP	2	1.00	0.0792
15	5	S1	EXP	2	0.99	0.0902
15	10	S1	EXP	2	0.98	0.1058
15	15	S1	EXP	2	0.98	0.1282
15	20	S1	EXP	2	0.98	0.1553
20	0	S1	EXP	2	1.00	0.0872
20	5	S1	EXP	2	1.00	0.0967
20	10	S1	EXP	2	1.00	0.1124
20	15	S1	EXP	2	1.00	0.1352
20	20	S1	EXP	2	1.00	0.1539
10	0	S1	EXP	3	0.96	0.0470
10	5	S1	EXP	3	0.96	0.0583
10	10	S1	EXP	3	0.95	0.0719

Table 5.7: (continued)

10	15	S1	EXP	3	0.95	0.0849
10	20	S1	EXP	3	0.91	0.1038
15	0	S1	EXP	3	0.98	0.0546
15	5	S1	EXP	3	0.97	0.0610
15	10	S1	EXP	3	0.98	0.0728
15	15	S1	EXP	3	0.97	0.0859
15	20	S1	EXP	3	0.96	0.1019
20	0	S1	EXP	3	1.00	0.0588
20	5	S1	EXP	3	1.00	0.0688
20	10	S1	EXP	3	1.00	0.0790
20	15	S1	EXP	3	1.00	0.0913
20	20	S1	EXP	3	1.00	0.1040
15	0	S1	EXP	4	0.99	0.0483
15	5	S1	EXP	4	0.99	0.0575
15	10	S1	EXP	4	0.97	0.0729
15	15	S1	EXP	4	0.96	0.0949
15	20	S1	EXP	4	0.96	0.1015
20	0	S1	EXP	4	0.98	0.0556
20	5	S1	EXP	4	1.00	0.0639
20	10	S1	EXP	4	0.98	0.0796
20	15	S1	EXP	4	0.99	0.0932
20	20	S1	EXP	4	0.99	0.1046

Table 5.8: Negative bag effect on EMDD-RL for the four-room environment. For different k values of the setting with strategy S1 and exponential model, the accuracy performances are depicted.

EMDD-RL (S1, EXP) Negative Bag Effect (four-room)						
Pstv	Ngtv	Strg	Model	Skip	Accuracy	Time
15	0	S1	EXP	1	0.84	0.2748
15	5	S1	EXP	1	0.89	0.3221
15	10	S1	EXP	1	0.91	0.4123
15	15	S1	EXP	1	0.91	0.4430
15	20	S1	EXP	1	0.90	0.5179
10	0	S1	EXP	2	0.80	0.1371
10	5	S1	EXP	2	0.84	0.1541
10	10	S1	EXP	2	0.83	0.1921
10	15	S1	EXP	2	0.84	0.2272
10	20	S1	EXP	2	0.85	0.2754
20	0	S1	EXP	3	0.91	0.1050
20	5	S1	EXP	3	0.89	0.1128
20	10	S1	EXP	3	0.93	0.1353
20	15	S1	EXP	3	0.91	0.1510
20	20	S1	EXP	3	0.90	0.1715
5	0	S1	EXP	4	0.60	0.0517
5	5	S1	EXP	4	0.72	0.0836
5	10	S1	EXP	4	0.65	0.1184
5	15	S1	EXP	4	0.57	0.1556
5	20	S1	EXP	4	0.54	0.1843

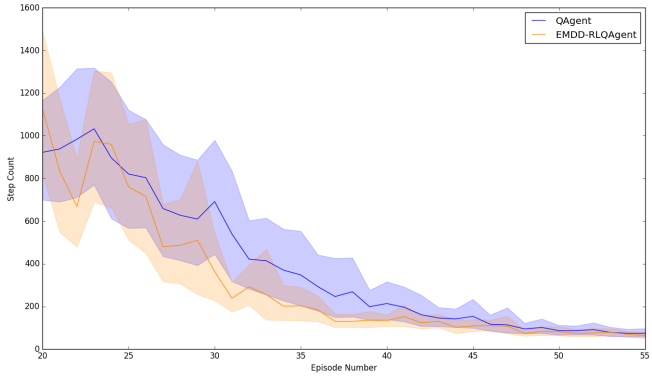


Figure 5.7: The total number of steps that the agents have taken to reach the goal state per episode is shown. The average step value is obtained from 40 experiments. The shaded areas represent the 95% bootstrap confidence interval.

5.2.8 Option Creation with EMDD-RL

In order to be compatible with SDD study, we have set up a separate experiment to show the usefulness of the EMDD-RL as a subgoal identifier for RL problems, we have compared its problem solving performance with a Q-learning agent in the two-room environment. By adhering to the work SDD [5] we have made use of the options framework and subgoal selection strategy described in the study. Only one option is permitted for the environment. The agent begins seeking a subgoal after gathering 20 episodes (all have been labeled as positive, no negative bag is created.). Afterwards, it updates the average value of detected subgoal candidates during upcoming episodes. If the average value of any candidate exceeds the threshold value (λ, we have set it as 0.6), it is picked as a subgoal. The option is created by inspecting the agent's trajectory history. After option creation, the learning method switches from Q-learning to Intra-option Value Learning [33]. The agent starts every episode from the state at the top-left corner of the environment and the states around the starting and goal states have been excluded for EMDD-RL calculations. The same action noise and Q-learning parameters of the data gathering Q-learning agent used in the experiment part are considered. This experiment has been repeated 40 times to obtain statistically reliable results for the subgoal identification performance on the problem.

In Figure 5.7, the agent creates the option between episodes 27 and 44 (From run to run the subgoal identification procedure might last shorter or longer, this is the reason

why the subgoals have been identified in different time steps). The option created helps the agent to attain the goal state with fewer steps compared to the Q-learning agent after 27^{th} episode till approximately 50^{th} episode.

CHAPTER 6

DISCUSSION AND FUTURE WORK

In this section, we elaborate and discuss the advantages and disadvantages of the adapted EMDD-RL algorithm that this thesis describes. The algorithm runs significantly faster than the SDD algorithm on the tested problems. Different from the original EMDD algorithm which surpassed the DD algorithm in both accuracy and speed, the EMDD-RL is superior or equal to its predecessor based on the DD algorithm in accuracy. The main disadvantage of the EMDD-RL is that it requires its parameters to be tweaked for the problem at hand. It possesses three hyperparameters: instance selecting strategies (S1, S2), bag label probability distribution models (linear, exponential), and the skipping factor for shrinking the initiation seed set. The initiation set selection method can easily be replaced with another selection method that does not require a parameter to be adjusted. But the former two need to be decided in advance. But depending on the requirements of the applications, the advantage demanded from the algorithm might be different. For the subgoal identification problem, we can name two important performance metrics: accuracy and speed. The EMDD-RL does not have a winning hyperparameter setting for accuracy performance. But for the speed performance, usage of the exponential model (leads to a fewer number of looping) and the strategy of selecting among selected positive instances (S1) has yielded the fastest configuration among all configurations and for all datasets we have tested. Our proposed selection strategy (S2) has degraded the speed performance of EMDD-RL compared to the original selection method (S1) but it has been able to deliver more accurate results on several datasets. Hence if the application considers the speed metric more, the EMDD-RL with strategy S1 and the exponential model for the label probabilities delivers the fastest execution time. On the other hand, for accuracy, a parameter search over the mentioned hyperparameters should be carried out.

The main advantage delivered by the original EMDD algorithm is that it enables us to reduce the initial test instances hence fewer instances are considered for the calculations compared to the DD algorithm for DD value maximization. We have altered the original initial test set selection method of the original EMDD algorithm. Since the grid-world problems exhibit local DD value similarities, it is quite reasonable to eliminate several states over the neighborhood for testing. Our strategy is to pick the smallest positive bag (since the global peak must occur in one of these states) and rule out some states and test the remaining for the maximum DD value. A "better" strategy to shrink the initial test set more will improve the speed performance of EMDD-RL further since fewer instances will have been used for the calculations and the computation overhead is reduced. A group of candidate methods can be devised depending on the graph or count-based metrics for the selection procedure of the initial test set.

Another important advantage of the EMDD-RL is that it is more resilient to large state space sizes. When the state space for the environment enlarges, the gap between speed performances of the EMDD-RL and SDD algorithms expands. Since the SDD algorithm checks all positive instances, it has to deal with far more instances compared to the EMDD-RL. Our state effect experiment results verify the expectation of the performance difference between algorithms.

An important future research direction for the thesis seems to be identifying the subgoals in Partially Observable Markov Decision Process (POMDP) problems. The literature for subgoal identification on these types of problems has not been matured yet. Some of the MDP subgoal identification methods might be attempted to be used but these methods face a serious problem emerging due to the very nature of the POMDP: uncertainty in the state information and state aliasing problems. In POMDP, the agent is not provided with the full state information similar to MDP, instead, the agent perceives some observations. The agent itself has to figure out or construct the necessary state information. With this uncertainty in the state information, most of the methods of MDP for the subgoal identification becomes useless since they strongly rely on the state information. It becomes very difficult to trust the calculated values by the count-based methods. For instance, the relative novelty metric will not be able to identify the gateway locations because the previously seen observation may be seen in the new room when the agent transitions through a gateway due to the fact that states in

different rooms may share the same observation semantic. Similarly, the graph-based methods suffer from the same problem since the distance and neighborhood information is obscured by again the uncertainty caused by the observation semantics. The SDD was shown to be useful for the identification of the subgoals in several POMDP problems [47]. Both the SDD and EMDD-RL make use of a transition graph that is degenerate due to the uncertainty. Since the SDD algorithm was shown to be performing well, we believe, by relying on the results obtained in this study, that the EMDD-RL may be more accurate and faster for subgoal identification in POMDP problems.

CHAPTER 7

CONCLUSION

For subgoal identification on reinforcement learning problems, we have adapted a multi-instance learning method called Expectation-Maximization Diverse Density (EMDD) that was shown to be superior to the Diverse Density algorithm in both speed and accuracy performances in the multi-instance learning domain. The EMDD algorithm first starts with an initial guess about the maximum DD valued state and gradually updates its guess by employing the Expectation-Maximization technique. We have tested its hyperparameters against speed and accuracy performances on two grid-world problems and compared it with its predecessor called SDD. Our speed results have shown a very significant speed gain against the SDD algorithm. The most accurate configuration for EMDD-RL has slightly changed from dataset to dataset (the configuration with exponential model and search strategy S1 has been superior on most of the datasets), but the fastest configuration has been obtained with the exponential model and the original selection strategy for all datasets and configurations. In both environments, the EMDD-RL has surpassed the SDD algorithm in speed and attained better or similar results in accuracy performance.

One of the big advantages of the EMDD-RL is that it makes use of fewer instances compared to the SDD algorithm and attains faster execution times without sacrificing the accuracy. The second advantage is that as the state space grows, the effect of state-space size on EMDD-RL is quite small compared to the SDD algorithm. As the state space enlarges, the SDD algorithm has to seek the peak DD value among more instances than EMDD-RL hence the speed performance gap between them further expands.

We believe our method described in this thesis can still be improved and employed

as a new method in other research areas. As an important future research direction, further development could take place in the initial set selection part of the EMDD-RL algorithm. As a simple strategy, we have populated the initial seed set from the shortest episode history of the agent. Other techniques that will shrink the set without missing the possible subgoal candidates will speed EMDD-RL further since the algorithm will be performing fewer calculations. Another possible strong research direction is that the EMDD-RL could be introduced to the Partially Observable MDP (POMDP) research area as a new subgoal identification method for POMDP problems.

REFERENCES

[1] Q. Zhang and S. A. Goldman, "Em-dd: An improved multiple-instance learning technique," in *Advances in Neural Information Processing Systems 14* (T. G. Dietterich, S. Becker, and Z. Ghahramani, eds.), pp. 1073–1080, MIT Press, 2002.

[2] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, May 1992.

[3] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, p. 181–211, Aug. 1999.

[4] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Mach. Learn.*, vol. 8, p. 293–321, May 1992.

[5] A. McGovern and A. G. Barto, "Automatic discovery of subgoals in reinforcement learning using diverse density," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, (San Francisco, CA, USA), p. 361–368, Morgan Kaufmann Publishers Inc., 2001.

[6] O. Şimşek, A. P. Wolfe, and A. G. Barto, "Identifying useful subgoals in reinforcement learning by local graph partitioning," in *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, (New York, NY, USA), p. 816–823, Association for Computing Machinery, 2005.

[7] I. Menache, S. Mannor, and N. Shimkin, "Q-cut - dynamic discovery of sub-goals in reinforcement learning," in *Proceedings of the 13th European Conference on Machine Learning*, ECML '02, (Berlin, Heidelberg), p. 295–306, Springer-Verlag, 2002.

[8] O. Şimşek and A. G. Barto, "Using relative novelty to identify useful temporal abstractions in reinforcement learning," in *Proceedings of the Twenty-First*

International Conference on Machine Learning, ICML '04, (New York, NY, USA), p. 95, Association for Computing Machinery, 2004.

[9] R. Kretchmar, T. Feil, and R. Bansal, "Improved automatic discovery of subgoals for options in hierarchical reinforcement learning," *Journal of Computer Science and Technology - JCST*, 01 2003.

[10] B. Ghazanfari, F. Afghah, and M. E. Taylor, "Sequential association rule mining for autonomously extracting hierarchical task structures in reinforcement learning," *IEEE Access*, vol. 8, pp. 11782–11799, 2020.

[11] O. Maron and T. Lozano-Pérez, "A framework for multiple-instance learning," in *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*, NIPS '97, (Cambridge, MA, USA), p. 570–576, MIT Press, 1998.

[12] R. BELLMAN, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[13] R. E. BELLMAN and S. E. DREYFUS, *Applied Dynamic Programming*. Princeton University Press, 1962.

[14] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, p. 9–44, Aug. 1988.

[15] G. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," *Technical Report CUED/F-INFENG/TR 166*, 11 1994.

[16] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, pp. 400–407, 09 1951.

[17] H. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems* (J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, eds.), vol. 23, pp. 2613–2621, Curran Associates, Inc., 2010.

[18] G. Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural Comput.*, vol. 6, p. 215–219, Mar. 1994.

[19] M. Riedmiller, "Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method," vol. 3720, pp. 317–328, 10 2005.

[20] J. García, I. López-Bueno, F. Fernández, and D. Borrajo, *A comparative study of discretization approaches for state space generalization in the keepaway soccer task*, pp. 161–190. 01 2011.

[21] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997.

[22] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359 – 366, 1989.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–33, 02 2015.

[24] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems* (S. Solla, T. Leen, and K. Müller, eds.), vol. 12, pp. 1057–1063, MIT Press, 2000.

[25] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, p. 229–256, May 1992.

[26] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2016.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 07 2017.

[28] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, p. 1889–1897, JMLR.org, 2015.

[29] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," 09 2017.

[30] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[31] S. S. Haykin, *Neural networks and learning machines*. Upper Saddle River, NJ: Pearson Education, third ed., 2009.

[32] A. Mcgovern, R. Sutton, and A. Fagg, "Roles of macro-actions in accelerating reinforcement learning," 02 1999.

[33] R. S. Sutton, D. Precup, and S. P. Singh, "Intra-option learning about temporally abstract actions," in *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, (San Francisco, CA, USA), p. 556–564, Morgan Kaufmann Publishers Inc., 1998.

[34] O. Şimşek and A. Barto, "Skill characterization based on betweenness," *Advances in Neural Information Processing Systems 21 - Proceedings of the 2008 Conference*, pp. 1497–1504, 01 2009.

[35] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez, "Solving the multiple instance problem with axis-parallel rectangles," *Artif. Intell.*, vol. 89, p. 31–71, Jan. 1997.

[36] J. Foulds and E. Frank, "A review of multi-instance learning assumptions," *The Knowledge Engineering Review*, vol. 25, no. 1, p. 1–25, 2010.

[37] F. Herrera, S. Ventura, R. Bello, C. Cornelis, A. Zafra, D. Sánchez-Tarragó, and S. Vluymans, *Multiple instance learning : foundations and algorithms*. Springer, 2016.

[38] X. Xu and E. Frank, "Logistic regression and boosting for labeled bags of instances," in *Advances in Knowledge Discovery and Data Mining* (H. Dai, R. Srikant, and C. Zhang, eds.), (Berlin, Heidelberg), pp. 272–281, Springer Berlin Heidelberg, 2004.

[39] Yixin Chen, Jinbo Bi, and J. Z. Wang, "Miles: Multiple-instance learning via

embedded instance selection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 1931–1947, 2006.

[40] W. Li and D. Y. yeung, "Mild: Multiple-instance learning via disambiguation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 1, pp. 76–89, 2010.

[41] J. Wang and J.-D. Zucker, "Solving multiple-instance problem: A lazy learning approach," 2000.

[42] S. Kazemitabar and H. Beigy, "Automatic discovery of subgoals in reinforcement learning using strongly connected components," vol. 5506, pp. 829–834, 11 2008.

[43] O. Maron and T. Lozano-Perez, *Learning from Ambiguity*. PhD thesis, USA, 1998. AAI0599603.

[44] A. Mcgovern, R. S. Sutton, and A. H. Fagg, "Roles of macro-actions in accelerating reinforcement learning," in *In Grace Hopper Celebration of Women in Computing*, pp. 13–18, 1997.

[45] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[46] C. F. J. Wu, "On the convergence properties of the em algorithm," *The Annals of Statistics*, vol. 11, no. 1, pp. 95–103, 1983.

[47] A. Demir, E. Çilden, and F. Polat, "A concept filtering approach for diverse density to discover subgoals in reinforcement learning," in *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 1–5, 2017.

APPENDIX A

Table A.3: EMDD-RL accuracy experimentation results on two-room. Run time is in seconds.

EMDD-RL Accuracy Experiment (two-room)							
Positive	Negative	Strategy	Model	Skip Factor	Average Loop	Accuracy	Run Time
5	0	S2	LIN	1	71.53	0.84	0.2097
5	5	S2	LIN	1	82.18	0.71	0.5154
5	10	S2	LIN	1	88.42	0.70	0.7276
5	15	S2	LIN	1	87.31	0.60	0.8851
5	20	S2	LIN	1	87.02	0.56	1.0684
10	0	S2	LIN	1	60.35	0.98	0.3224
10	5	S2	LIN	1	70.54	0.97	0.6176
10	10	S2	LIN	1	78.54	0.96	0.8454
10	15	S2	LIN	1	78.39	0.93	1.0544
10	20	S2	LIN	1	78.22	0.93	1.2302
15	0	S2	LIN	1	49.87	1.00	0.3601
15	5	S2	LIN	1	59.77	0.98	0.6249
15	10	S2	LIN	1	68.66	0.98	0.8997
15	15	S2	LIN	1	69.11	0.98	1.0459
15	20	S2	LIN	1	68.2	0.98	1.1831
20	0	S2	LIN	1	45.33	1.00	0.4182
20	5	S2	LIN	1	53.88	1.00	0.6506
20	10	S2	LIN	1	63.45	1.00	0.9049
20	15	S2	LIN	1	63.55	1.00	1.0634
20	20	S2	LIN	1	63.54	1.00	1.2011
5	0	S2	LIN	2	36.44	0.86	0.1162

Table A.3: (continued)

5	5	S2	LIN	2	41.66	0.73	0.2720
5	10	S2	LIN	2	44.93	0.69	0.3992
5	15	S2	LIN	2	44.43	0.60	0.4690
5	20	S2	LIN	2	44.56	0.57	0.5607
10	0	S2	LIN	2	30.74	0.99	0.1758
10	5	S2	LIN	2	36.1	0.96	0.3178
10	10	S2	LIN	2	40.11	0.93	0.4528
10	15	S2	LIN	2	39.84	0.91	0.5447
10	20	S2	LIN	2	39.49	0.89	0.6219
15	0	S2	LIN	2	25.72	1.00	0.1923
15	5	S2	LIN	2	30.69	0.98	0.3333
15	10	S2	LIN	2	35.19	0.98	0.4539
15	15	S2	LIN	2	35.21	0.97	0.5535
15	20	S2	LIN	2	34.62	0.97	0.6122
20	0	S2	LIN	2	23.38	1.00	0.2092
20	5	S2	LIN	2	27.81	1.00	0.3446
20	10	S2	LIN	2	32.76	1.00	0.4785
20	15	S2	LIN	2	32.63	1.00	0.5673
20	20	S2	LIN	2	32.47	1.00	0.6354
5	0	S2	LIN	3	24.3	0.77	0.0791
5	5	S2	LIN	3	28.03	0.65	0.1809
5	10	S2	LIN	3	30.07	0.69	0.2666
5	15	S2	LIN	3	29.68	0.62	0.3311
5	20	S2	LIN	3	29.69	0.55	0.3990
10	0	S2	LIN	3	20.46	0.96	0.1120
10	5	S2	LIN	3	23.95	0.94	0.2204
10	10	S2	LIN	3	27.02	0.95	0.3210
10	15	S2	LIN	3	26.88	0.90	0.3832
10	20	S2	LIN	3	26.84	0.87	0.4318
15	0	S2	LIN	3	16.93	0.99	0.1336
15	5	S2	LIN	3	20.34	0.98	0.2260
15	10	S2	LIN	3	23.54	0.97	0.3178
15	15	S2	LIN	3	23.7	0.98	0.3665
15	20	S2	LIN	3	23.49	0.97	0.4184

Table A.3: (continued)

20	0	S2	LIN	3	15.47	1.00	0.1450
20	5	S2	LIN	3	18.37	1.00	0.2371
20	10	S2	LIN	3	21.74	1.00	0.3351
20	15	S2	LIN	3	21.84	1.00	0.4149
20	20	S2	LIN	3	21.96	1.00	0.4339
5	0	S2	LIN	4	18.82	0.82	0.0618
5	5	S2	LIN	4	21.58	0.75	0.1453
5	10	S2	LIN	4	23.18	0.70	0.2238
5	15	S2	LIN	4	23.03	0.61	0.2836
5	20	S2	LIN	4	23.06	0.60	0.3390
10	0	S2	LIN	4	15.83	0.98	0.0962
10	5	S2	LIN	4	18.57	0.95	0.1935
10	10	S2	LIN	4	20.35	0.91	0.2600
10	15	S2	LIN	4	20.33	0.87	0.3140
10	20	S2	LIN	4	19.97	0.86	0.3623
15	0	S2	LIN	4	13.26	0.99	0.1094
15	5	S2	LIN	4	16.03	0.98	0.1964
15	10	S2	LIN	4	18.24	0.97	0.2675
15	15	S2	LIN	4	18.27	0.96	0.3296
15	20	S2	LIN	4	17.85	0.95	0.3719
20	0	S2	LIN	4	11.97	0.99	0.1244
20	5	S2	LIN	4	14.33	1.00	0.2067
20	10	S2	LIN	4	16.88	1.00	0.2828
20	15	S2	LIN	4	16.91	1.00	0.3311
20	20	S2	LIN	4	16.66	0.99	0.3706
5	0	S1	LIN	1	59.39	0.78	0.0955
5	5	S1	LIN	1	69.01	0.70	0.1713
5	10	S1	LIN	1	74.73	0.70	0.2460
5	15	S1	LIN	1	74.4	0.59	0.3229
5	20	S1	LIN	1	72.94	0.56	0.3757
10	0	S1	LIN	1	53.56	0.98	0.1402
10	5	S1	LIN	1	64.22	0.97	0.2112
10	10	S1	LIN	1	69.27	0.96	0.2813
10	15	S1	LIN	1	68.39	0.93	0.3427

Table A.3: (continued)

10	20	S1	LIN	1	67.05	0.93	0.3992
15	0	S1	LIN	1	46.01	1.00	0.1653
15	5	S1	LIN	1	57.68	0.99	0.2287
15	10	S1	LIN	1	64.46	0.98	0.3038
15	15	S1	LIN	1	64.44	0.98	0.3565
15	20	S1	LIN	1	61.94	0.98	0.4142
20	0	S1	LIN	1	42.89	1.00	0.1869
20	5	S1	LIN	1	53.64	1.00	0.2548
20	10	S1	LIN	1	60.55	1.00	0.3374
20	15	S1	LIN	1	60.76	1.00	0.3807
20	20	S1	LIN	1	59.5	1.00	0.4371
5	0	S1	LIN	2	30.33	0.78	0.0527
5	5	S1	LIN	2	35.03	0.70	0.0899
5	10	S1	LIN	2	37.8	0.71	0.1404
5	15	S1	LIN	2	37.6	0.57	0.1646
5	20	S1	LIN	2	36.64	0.53	0.1950
10	0	S1	LIN	2	27.23	0.99	0.0739
10	5	S1	LIN	2	32.59	0.95	0.1086
10	10	S1	LIN	2	35.26	0.93	0.1535
10	15	S1	LIN	2	34.53	0.91	0.1864
10	20	S1	LIN	2	33.7	0.90	0.2142
15	0	S1	LIN	2	23.79	1.00	0.0851
15	5	S1	LIN	2	29.87	0.98	0.1237
15	10	S1	LIN	2	33.23	0.98	0.1682
15	15	S1	LIN	2	32.87	0.97	0.1971
15	20	S1	LIN	2	31.58	0.97	0.2224
20	0	S1	LIN	2	22.01	1.00	0.0996
20	5	S1	LIN	2	27.77	1.00	0.1412
20	10	S1	LIN	2	31.19	1.00	0.1794
20	15	S1	LIN	2	31.12	1.00	0.2115
20	20	S1	LIN	2	30.24	1.00	0.2376
5	0	S1	LIN	3	20.45	0.76	0.0352
5	5	S1	LIN	3	24.0	0.63	0.0669
5	10	S1	LIN	3	25.93	0.69	0.0919

Table A.3: (continued)

5	15	S1	LIN	3	25.68	0.58	0.1183
5	20	S1	LIN	3	25.06	0.55	0.1407
10	0	S1	LIN	3	18.1	0.96	0.0509
10	5	S1	LIN	3	21.76	0.94	0.0787
10	10	S1	LIN	3	23.55	0.95	0.1092
10	15	S1	LIN	3	23.12	0.89	0.1268
10	20	S1	LIN	3	22.78	0.89	0.1499
15	0	S1	LIN	3	15.68	0.98	0.0562
15	5	S1	LIN	3	19.57	0.99	0.0875
15	10	S1	LIN	3	21.99	0.97	0.1147
15	15	S1	LIN	3	21.85	0.97	0.1349
15	20	S1	LIN	3	20.96	0.96	0.1542
20	0	S1	LIN	3	14.52	1.00	0.0646
20	5	S1	LIN	3	18.25	1.00	0.0938
20	10	S1	LIN	3	20.81	1.00	0.1364
20	15	S1	LIN	3	20.77	1.00	0.1458
20	20	S1	LIN	3	20.27	1.00	0.1644
5	0	S1	LIN	4	15.54	0.77	0.0266
5	5	S1	LIN	4	18.13	0.74	0.0528
5	10	S1	LIN	4	19.62	0.78	0.0854
5	15	S1	LIN	4	19.44	0.63	0.1142
5	20	S1	LIN	4	19.02	0.59	0.1411
10	0	S1	LIN	4	13.92	0.96	0.0424
10	5	S1	LIN	4	16.85	0.94	0.0712
10	10	S1	LIN	4	18.19	0.91	0.1022
10	15	S1	LIN	4	17.64	0.86	0.1276
10	20	S1	LIN	4	17.05	0.85	0.1615
15	0	S1	LIN	4	12.26	0.98	0.0521
15	5	S1	LIN	4	15.58	0.98	0.0864
15	10	S1	LIN	4	17.06	0.97	0.1162
15	15	S1	LIN	4	17.05	0.95	0.1347
15	20	S1	LIN	4	16.26	0.93	0.1575
20	0	S1	LIN	4	11.18	0.99	0.0606
20	5	S1	LIN	4	14.27	1.00	0.0965

Table A.3: (continued)

20	10	S1	LIN	4	16.14	1.00	0.1267
20	15	S1	LIN	4	15.98	1.00	0.1452
20	20	S1	LIN	4	15.47	0.99	0.1661
5	0	S2	EXP	1	60.59	0.84	0.1366
5	5	S2	EXP	1	52.24	0.71	0.2307
5	10	S2	EXP	1	49.0	0.70	0.3036
5	15	S2	EXP	1	49.69	0.63	0.4011
5	20	S2	EXP	1	49.99	0.59	0.4915
10	0	S2	EXP	1	51.35	0.98	0.1976
10	5	S2	EXP	1	44.2	0.97	0.2813
10	10	S2	EXP	1	42.48	0.95	0.3480
10	15	S2	EXP	1	43.05	0.94	0.4296
10	20	S2	EXP	1	44.2	0.93	0.5185
15	0	S2	EXP	1	44.2	1.00	0.2477
15	5	S2	EXP	1	37.83	0.98	0.3036
15	10	S2	EXP	1	35.86	0.99	0.3666
15	15	S2	EXP	1	36.93	0.99	0.4303
15	20	S2	EXP	1	37.62	0.97	0.5104
20	0	S2	EXP	1	40.14	1.00	0.2718
20	5	S2	EXP	1	34.88	1.00	0.3289
20	10	S2	EXP	1	33.37	1.00	0.3934
20	15	S2	EXP	1	34.37	1.00	0.4474
20	20	S2	EXP	1	35.12	1.00	0.5244
5	0	S2	EXP	2	30.73	0.86	0.0740
5	5	S2	EXP	2	26.5	0.73	0.1209
5	10	S2	EXP	2	24.75	0.67	0.1612
5	15	S2	EXP	2	24.91	0.66	0.1998
5	20	S2	EXP	2	25.13	0.63	0.2489
10	0	S2	EXP	2	25.76	0.99	0.1056
10	5	S2	EXP	2	22.12	0.96	0.1429
10	10	S2	EXP	2	21.37	0.95	0.1804
10	15	S2	EXP	2	21.76	0.95	0.2266
10	20	S2	EXP	2	22.5	0.92	0.2658
15	0	S2	EXP	2	22.43	1.00	0.1277

Table A.3: (continued)

15	5	S2	EXP	2	19.12	0.99	0.1531
15	10	S2	EXP	2	18.35	0.98	0.1895
15	15	S2	EXP	2	18.81	0.98	0.2197
15	20	S2	EXP	2	19.27	0.98	0.2629
20	0	S2	EXP	2	20.49	1.00	0.1445
20	5	S2	EXP	2	17.63	1.00	0.1743
20	10	S2	EXP	2	17.01	1.00	0.2032
20	15	S2	EXP	2	17.58	1.00	0.2323
20	20	S2	EXP	2	18.09	1.00	0.2694
5	0	S2	EXP	3	20.86	0.78	0.0504
5	5	S2	EXP	3	17.85	0.69	0.0839
5	10	S2	EXP	3	16.64	0.74	0.1073
5	15	S2	EXP	3	16.89	0.69	0.1353
5	20	S2	EXP	3	16.98	0.66	0.1658
10	0	S2	EXP	3	17.43	0.96	0.0740
10	5	S2	EXP	3	15.19	0.97	0.0984
10	10	S2	EXP	3	14.7	0.96	0.1245
10	15	S2	EXP	3	14.81	0.95	0.1520
10	20	S2	EXP	3	15.36	0.91	0.1807
15	0	S2	EXP	3	15.17	0.99	0.0845
15	5	S2	EXP	3	13.18	0.98	0.1098
15	10	S2	EXP	3	12.45	0.98	0.1285
15	15	S2	EXP	3	12.96	0.97	0.1591
15	20	S2	EXP	3	13.09	0.96	0.1819
20	0	S2	EXP	3	13.82	1.00	0.0938
20	5	S2	EXP	3	12.09	1.00	0.1132
20	10	S2	EXP	3	11.56	1.00	0.1361
20	15	S2	EXP	3	11.78	1.00	0.1561
20	20	S2	EXP	3	12.03	1.00	0.1782
5	0	S2	EXP	4	16.04	0.83	0.0409
5	5	S2	EXP	4	13.57	0.81	0.0688
5	10	S2	EXP	4	12.67	0.76	0.0962
5	15	S2	EXP	4	12.77	0.76	0.1263
5	20	S2	EXP	4	12.86	0.65	0.1511

Table A.3: (continued)

10	0	S2	EXP	4	13.1	0.97	0.0617
10	5	S2	EXP	4	11.23	0.93	0.0845
10	10	S2	EXP	4	10.96	0.91	0.1093
10	15	S2	EXP	4	11.15	0.89	0.1394
10	20	S2	EXP	4	11.57	0.86	0.1627
15	0	S2	EXP	4	11.54	0.99	0.0752
15	5	S2	EXP	4	9.56	0.99	0.0934
15	10	S2	EXP	4	9.43	0.97	0.1131
15	15	S2	EXP	4	9.57	0.96	0.1369
15	20	S2	EXP	4	9.73	0.96	0.1628
20	0	S2	EXP	4	10.46	0.98	0.0832
20	5	S2	EXP	4	8.93	1.00	0.1017
20	10	S2	EXP	4	8.69	0.99	0.1209
20	15	S2	EXP	4	8.91	0.99	0.1481
20	20	S2	EXP	4	9.18	0.99	0.1753
5	0	S1	EXP	1	58.1	0.78	0.0935
5	5	S1	EXP	1	52.39	0.70	0.1460
5	10	S1	EXP	1	47.02	0.70	0.1862
5	15	S1	EXP	1	47.15	0.64	0.2356
5	20	S1	EXP	1	47.51	0.58	0.2872
10	0	S1	EXP	1	49.07	0.98	0.1322
10	5	S1	EXP	1	43.48	0.97	0.1637
10	10	S1	EXP	1	41.47	0.95	0.2042
10	15	S1	EXP	1	41.96	0.94	0.2530
10	20	S1	EXP	1	43.49	0.93	0.3054
15	0	S1	EXP	1	42.21	1.00	0.1513
15	5	S1	EXP	1	37.88	0.98	0.1812
15	10	S1	EXP	1	35.48	0.99	0.2194
15	15	S1	EXP	1	36.48	0.99	0.2492
15	20	S1	EXP	1	37.53	0.97	0.2937
20	0	S1	EXP	1	37.97	1.00	0.1703
20	5	S1	EXP	1	34.84	1.00	0.1921
20	10	S1	EXP	1	33.39	1.00	0.2213
20	15	S1	EXP	1	34.24	1.00	0.2687

Table A.3: (continued)

20	20	S1	EXP	1	35.01	1.00	0.3003
5	0	S1	EXP	2	29.62	0.78	0.0503
5	5	S1	EXP	2	26.61	0.70	0.0760
5	10	S1	EXP	2	23.77	0.67	0.0990
5	15	S1	EXP	2	23.76	0.66	0.1180
5	20	S1	EXP	2	23.93	0.62	0.1450
10	0	S1	EXP	2	24.78	0.99	0.0687
10	5	S1	EXP	2	21.73	0.95	0.0839
10	10	S1	EXP	2	20.82	0.95	0.1048
10	15	S1	EXP	2	21.26	0.95	0.1258
10	20	S1	EXP	2	22.18	0.92	0.1524
15	0	S1	EXP	2	21.42	1.00	0.0792
15	5	S1	EXP	2	19.24	0.99	0.0902
15	10	S1	EXP	2	18.17	0.98	0.1058
15	15	S1	EXP	2	18.6	0.98	0.1282
15	20	S1	EXP	2	19.23	0.98	0.1553
20	0	S1	EXP	2	19.39	1.00	0.0872
20	5	S1	EXP	2	17.67	1.00	0.0967
20	10	S1	EXP	2	17.04	1.00	0.1124
20	15	S1	EXP	2	17.53	1.00	0.1352
20	20	S1	EXP	2	18.02	1.00	0.1539
5	0	S1	EXP	3	20.05	0.76	0.0345
5	5	S1	EXP	3	18.0	0.69	0.0520
5	10	S1	EXP	3	16.0	0.71	0.0642
5	15	S1	EXP	3	16.1	0.67	0.0823
5	20	S1	EXP	3	16.19	0.65	0.0970
10	0	S1	EXP	3	16.85	0.96	0.0470
10	5	S1	EXP	3	15.08	0.96	0.0583
10	10	S1	EXP	3	14.35	0.95	0.0719
10	15	S1	EXP	3	14.57	0.95	0.0849
10	20	S1	EXP	3	15.23	0.91	0.1038
15	0	S1	EXP	3	14.52	0.98	0.0546
15	5	S1	EXP	3	13.17	0.97	0.0610
15	10	S1	EXP	3	12.34	0.98	0.0728

Table A.3: (continued)

15	15	S1	EXP	3	12.82	0.97	0.0859
15	20	S1	EXP	3	13.06	0.96	0.1019
20	0	S1	EXP	3	13.08	1.00	0.0588
20	5	S1	EXP	3	12.04	1.00	0.0688
20	10	S1	EXP	3	11.58	1.00	0.0790
20	15	S1	EXP	3	11.77	1.00	0.0913
20	20	S1	EXP	3	11.96	1.00	0.1040
5	0	S1	EXP	4	15.33	0.77	0.0271
5	5	S1	EXP	4	13.74	0.75	0.0459
5	10	S1	EXP	4	12.16	0.75	0.0599
5	15	S1	EXP	4	12.22	0.74	0.0819
5	20	S1	EXP	4	12.24	0.64	0.1027
10	0	S1	EXP	4	12.64	0.97	0.0413
10	5	S1	EXP	4	11.13	0.93	0.0533
10	10	S1	EXP	4	10.75	0.91	0.0694
10	15	S1	EXP	4	10.9	0.88	0.0975
10	20	S1	EXP	4	11.4	0.85	0.1032
15	0	S1	EXP	4	11.01	0.99	0.0483
15	5	S1	EXP	4	9.67	0.99	0.0575
15	10	S1	EXP	4	9.3	0.97	0.0729
15	15	S1	EXP	4	9.47	0.96	0.0949
15	20	S1	EXP	4	9.72	0.96	0.1015
20	0	S1	EXP	4	9.8	0.98	0.0556
20	5	S1	EXP	4	8.98	1.00	0.0639
20	10	S1	EXP	4	8.71	0.98	0.0796
20	15	S1	EXP	4	8.88	0.99	0.0932
20	20	S1	EXP	4	9.15	0.99	0.1046

Table A.1: SDD accuracy experimentation results on two-room. Run time is in seconds.

SDD Accuracy Experiment (two-room)				
Dataset	Positive	Negative	Accuracy	Run Time
STEPBALANCED	5	0	0.92	0.0724
STEPBALANCED	5	5	0.70	0.1365
STEPBALANCED	5	10	0.69	0.1971
STEPBALANCED	5	15	0.60	0.2618
STEPBALANCED	5	20	0.57	0.3271
STEPBALANCED	10	0	0.98	0.1591
STEPBALANCED	10	5	0.97	0.2268
STEPBALANCED	10	10	0.96	0.3039
STEPBALANCED	10	15	0.93	0.3939
STEPBALANCED	10	20	0.93	0.4649
STEPBALANCED	15	0	1.00	0.2386
STEPBALANCED	15	5	0.99	0.3123
STEPBALANCED	15	10	0.98	0.3895
STEPBALANCED	15	15	0.98	0.4711
STEPBALANCED	15	20	0.98	0.5564
STEPBALANCED	20	0	1.00	0.3198
STEPBALANCED	20	5	1.00	0.3906
STEPBALANCED	20	10	1.00	0.4823
STEPBALANCED	20	15	1.00	0.5792
STEPBALANCED	20	20	1.00	0.6416

Table A.4: EMDD-RL accuracy experimentation results on four-room. Run time is in seconds.

EMDD-RL Accuracy Experiment (four-room)							
Positive	Negative	Strategy	Model	Skip Factor	Average Loop	Accuracy	Run Time

Table A.4: (continued)

5	0	S2	LIN	1	97.91	0.58	0.5019
5	5	S2	LIN	1	108.29	0.60	1.1218
5	10	S2	LIN	1	111.84	0.60	1.5874
5	15	S2	LIN	1	112.92	0.59	2.0944
5	20	S2	LIN	1	111.44	0.59	2.2964
10	0	S2	LIN	1	76.12	0.84	0.8015
10	5	S2	LIN	1	87.62	0.81	1.4093
10	10	S2	LIN	1	90.06	0.78	1.9013
10	15	S2	LIN	1	90.17	0.82	2.2250
10	20	S2	LIN	1	90.76	0.81	2.6630
15	0	S2	LIN	1	65.9	0.82	0.9893
15	5	S2	LIN	1	74.08	0.81	1.5982
15	10	S2	LIN	1	74.41	0.83	1.9912
15	15	S2	LIN	1	74.91	0.82	2.2867
15	20	S2	LIN	1	76.0	0.80	2.6725
20	0	S2	LIN	1	53.66	0.94	1.0460
20	5	S2	LIN	1	60.32	0.90	1.5868
20	10	S2	LIN	1	60.92	0.88	1.9730
20	15	S2	LIN	1	61.45	0.88	2.1825
20	20	S2	LIN	1	62.8	0.83	2.5855
5	0	S2	LIN	2	49.71	0.59	0.2667
5	5	S2	LIN	2	55.04	0.62	0.5881
5	10	S2	LIN	2	56.83	0.66	0.8124
5	15	S2	LIN	2	57.4	0.59	1.0144
5	20	S2	LIN	2	56.56	0.60	1.1747
10	0	S2	LIN	2	38.51	0.84	0.4229
10	5	S2	LIN	2	44.41	0.83	0.7665
10	10	S2	LIN	2	45.63	0.80	1.0433
10	15	S2	LIN	2	45.46	0.82	1.2007
10	20	S2	LIN	2	45.75	0.78	1.3903
15	0	S2	LIN	2	33.29	0.82	0.5167
15	5	S2	LIN	2	37.86	0.79	0.8544
15	10	S2	LIN	2	38.07	0.77	1.0455
15	15	S2	LIN	2	38.38	0.77	1.2090

Table A.4: (continued)

15	20	S2	LIN	2	38.79	0.75	1.3895
20	0	S2	LIN	2	27.43	0.91	0.5378
20	5	S2	LIN	2	31.08	0.87	0.8613
20	10	S2	LIN	2	31.33	0.87	1.0104
20	15	S2	LIN	2	31.64	0.87	1.1568
20	20	S2	LIN	2	32.08	0.85	1.3089
5	0	S2	LIN	3	33.43	0.61	0.1813
5	5	S2	LIN	3	37.17	0.62	0.3967
5	10	S2	LIN	3	38.45	0.67	0.5967
5	15	S2	LIN	3	38.96	0.60	0.7061
5	20	S2	LIN	3	38.35	0.62	0.8673
10	0	S2	LIN	3	26.33	0.85	0.3011
10	5	S2	LIN	3	30.49	0.82	0.5205
10	10	S2	LIN	3	31.24	0.80	0.6852
10	15	S2	LIN	3	31.06	0.82	0.8168
10	20	S2	LIN	3	31.2	0.79	0.9496
15	0	S2	LIN	3	22.5	0.83	0.3577
15	5	S2	LIN	3	25.27	0.83	0.5712
15	10	S2	LIN	3	25.23	0.83	0.7003
15	15	S2	LIN	3	25.52	0.81	0.8396
15	20	S2	LIN	3	25.76	0.76	0.9551
20	0	S2	LIN	3	18.21	0.90	0.3805
20	5	S2	LIN	3	20.85	0.90	0.5672
20	10	S2	LIN	3	21.16	0.89	0.6868
20	15	S2	LIN	3	21.18	0.89	0.8307
20	20	S2	LIN	3	21.46	0.86	0.9057
5	0	S2	LIN	4	25.22	0.60	0.1457
5	5	S2	LIN	4	28.24	0.70	0.3270
5	10	S2	LIN	4	29.18	0.70	0.4752
5	15	S2	LIN	4	29.54	0.61	0.6022
5	20	S2	LIN	4	29.09	0.60	0.7451
10	0	S2	LIN	4	19.57	0.85	0.2380
10	5	S2	LIN	4	22.82	0.81	0.4311
10	10	S2	LIN	4	23.61	0.79	0.5766

Table A.4: (continued)

10	15	S2	LIN	4	23.42	0.81	0.6754
10	20	S2	LIN	4	23.72	0.78	0.7951
15	0	S2	LIN	4	16.86	0.83	0.2911
15	5	S2	LIN	4	19.15	0.77	0.4700
15	10	S2	LIN	4	19.28	0.76	0.6718
15	15	S2	LIN	4	19.56	0.75	0.6948
15	20	S2	LIN	4	19.87	0.71	0.8014
20	0	S2	LIN	4	13.94	0.86	0.3066
20	5	S2	LIN	4	15.94	0.85	0.4718
20	10	S2	LIN	4	16.07	0.83	0.6393
20	15	S2	LIN	4	16.31	0.83	0.7141
20	20	S2	LIN	4	16.42	0.82	0.7639
5	0	S1	LIN	1	77.64	0.56	0.1928
5	5	S1	LIN	1	90.59	0.55	0.3477
5	10	S1	LIN	1	93.38	0.55	0.4924
5	15	S1	LIN	1	93.58	0.55	0.6047
5	20	S1	LIN	1	92.15	0.56	0.7163
10	0	S1	LIN	1	67.56	0.82	0.2572
10	5	S1	LIN	1	77.38	0.82	0.3812
10	10	S1	LIN	1	77.38	0.80	0.4805
10	15	S1	LIN	1	77.84	0.81	0.5812
10	20	S1	LIN	1	77.79	0.79	0.6609
15	0	S1	LIN	1	61.48	0.86	0.2987
15	5	S1	LIN	1	68.75	0.82	0.4025
15	10	S1	LIN	1	69.76	0.83	0.5054
15	15	S1	LIN	1	68.84	0.81	0.6024
15	20	S1	LIN	1	69.05	0.80	0.7070
20	0	S1	LIN	1	50.89	0.92	0.3192
20	5	S1	LIN	1	59.58	0.86	0.4221
20	10	S1	LIN	1	60.32	0.86	0.4970
20	15	S1	LIN	1	58.48	0.86	0.5713
20	20	S1	LIN	1	58.45	0.82	0.6430
5	0	S1	LIN	2	39.26	0.59	0.0984
5	5	S1	LIN	2	39.26	0.58	0.0984

Table A.4: (continued)

5	10	S1	LIN	2	47.2	0.62	0.2512
5	15	S1	LIN	2	47.18	0.56	0.3087
5	20	S1	LIN	2	46.57	0.56	0.3936
10	0	S1	LIN	2	33.91	0.82	0.1381
10	5	S1	LIN	2	39.3	0.83	0.2059
10	10	S1	LIN	2	39.23	0.82	0.2565
10	15	S1	LIN	2	39.47	0.80	0.3064
10	20	S1	LIN	2	39.23	0.78	0.3640
15	0	S1	LIN	2	30.74	0.85	0.1620
15	5	S1	LIN	2	34.64	0.83	0.2286
15	10	S1	LIN	2	35.25	0.80	0.3002
15	15	S1	LIN	2	34.99	0.79	0.3229
15	20	S1	LIN	2	34.93	0.77	0.3731
20	0	S1	LIN	2	26.0	0.90	0.1699
20	5	S1	LIN	2	30.47	0.84	0.2276
20	10	S1	LIN	2	30.91	0.85	0.2730
20	15	S1	LIN	2	29.97	0.85	0.3214
20	20	S1	LIN	2	29.9	0.82	0.3669
5	0	S1	LIN	3	26.19	0.61	0.0723
5	5	S1	LIN	3	30.74	0.60	0.1318
5	10	S1	LIN	3	31.94	0.58	0.1819
5	15	S1	LIN	3	32.07	0.53	0.2184
5	20	S1	LIN	3	31.64	0.54	0.2670
10	0	S1	LIN	3	22.91	0.84	0.0962
10	5	S1	LIN	3	26.64	0.84	0.1463
10	10	S1	LIN	3	26.7	0.82	0.1851
10	15	S1	LIN	3	26.64	0.81	0.2171
10	20	S1	LIN	3	26.61	0.77	0.2582
15	0	S1	LIN	3	21.12	0.85	0.1120
15	5	S1	LIN	3	23.68	0.84	0.1616
15	10	S1	LIN	3	24.09	0.83	0.1960
15	15	S1	LIN	3	23.79	0.80	0.2246
15	20	S1	LIN	3	23.6	0.74	0.2611
20	0	S1	LIN	3	17.41	0.87	0.1207

Table A.4: (continued)

20	5	S1	LIN	3	20.49	0.84	0.1606
20	10	S1	LIN	3	20.84	0.85	0.1978
20	15	S1	LIN	3	20.06	0.83	0.2146
20	20	S1	LIN	3	19.87	0.79	0.2395
5	0	S1	LIN	4	20.09	0.63	0.0529
5	5	S1	LIN	4	23.55	0.61	0.1101
5	10	S1	LIN	4	24.07	0.61	0.1665
5	15	S1	LIN	4	24.17	0.55	0.2058
5	20	S1	LIN	4	23.78	0.53	0.2495
10	0	S1	LIN	4	17.3	0.83	0.0755
10	5	S1	LIN	4	19.9	0.82	0.1267
10	10	S1	LIN	4	20.0	0.81	0.1663
10	15	S1	LIN	4	20.09	0.80	0.1998
10	20	S1	LIN	4	19.95	0.78	0.2396
15	0	S1	LIN	4	15.81	0.87	0.0971
15	5	S1	LIN	4	17.43	0.81	0.1415
15	10	S1	LIN	4	17.93	0.77	0.1763
15	15	S1	LIN	4	17.86	0.75	0.2102
15	20	S1	LIN	4	17.83	0.72	0.2471
20	0	S1	LIN	4	13.27	0.87	0.1034
20	5	S1	LIN	4	15.69	0.82	0.1533
20	10	S1	LIN	4	16.03	0.80	0.1789
20	15	S1	LIN	4	15.68	0.79	0.2155
20	20	S1	LIN	4	15.64	0.79	0.2404
5	0	S2	EXP	1	80.74	0.54	0.2960
5	5	S2	EXP	1	61.53	0.55	0.4776
5	10	S2	EXP	1	60.53	0.58	0.6552
5	15	S2	EXP	1	61.43	0.57	0.7901
5	20	S2	EXP	1	61.89	0.61	0.9520
10	0	S2	EXP	1	65.27	0.85	0.4534
10	5	S2	EXP	1	52.13	0.81	0.5882
10	10	S2	EXP	1	51.63	0.81	0.7281
10	15	S2	EXP	1	53.61	0.84	0.8865
10	20	S2	EXP	1	54.53	0.86	1.0393

Table A.4: (continued)

15	0	S2	EXP	1	56.79	0.84	0.5487
15	5	S2	EXP	1	48.35	0.87	0.6583
15	10	S2	EXP	1	47.18	0.91	0.8087
15	15	S2	EXP	1	47.71	0.91	0.9524
15	20	S2	EXP	1	48.98	0.91	1.1150
20	0	S2	EXP	1	46.4	0.94	0.5626
20	5	S2	EXP	1	40.52	0.92	0.6642
20	10	S2	EXP	1	40.77	0.94	0.8021
20	15	S2	EXP	1	41.03	0.93	0.9512
20	20	S2	EXP	1	43.22	0.92	1.0708
5	0	S2	EXP	2	40.9	0.54	0.1575
5	5	S2	EXP	2	31.04	0.60	0.2466
5	10	S2	EXP	2	30.66	0.68	0.3321
5	15	S2	EXP	2	31.13	0.62	0.4132
5	20	S2	EXP	2	31.31	0.61	0.4824
10	0	S2	EXP	2	33.19	0.84	0.2362
10	5	S2	EXP	2	26.16	0.84	0.3069
10	10	S2	EXP	2	25.92	0.82	0.3893
10	15	S2	EXP	2	26.91	0.85	0.4660
10	20	S2	EXP	2	27.57	0.87	0.5526
15	0	S2	EXP	2	28.54	0.84	0.2805
15	5	S2	EXP	2	24.3	0.90	0.3501
15	10	S2	EXP	2	23.9	0.91	0.4200
15	15	S2	EXP	2	24.19	0.91	0.4851
15	20	S2	EXP	2	24.82	0.89	0.5709
20	0	S2	EXP	2	23.64	0.92	0.2953
20	5	S2	EXP	2	20.67	0.90	0.3467
20	10	S2	EXP	2	20.72	0.93	0.4248
20	15	S2	EXP	2	20.83	0.93	0.4774
20	20	S2	EXP	2	21.94	0.92	0.5676
5	0	S2	EXP	3	27.63	0.56	0.1113
5	5	S2	EXP	3	21.0	0.68	0.1814
5	10	S2	EXP	3	20.65	0.65	0.2380
5	15	S2	EXP	3	20.83	0.65	0.2796

Table A.4: (continued)

5	20	S2	EXP	3	21.02	0.66	0.3285
10	0	S2	EXP	3	22.43	0.86	0.1626
10	5	S2	EXP	3	17.84	0.85	0.2145
10	10	S2	EXP	3	17.6	0.84	0.2625
10	15	S2	EXP	3	18.33	0.86	0.3145
10	20	S2	EXP	3	18.62	0.82	0.3729
15	0	S2	EXP	3	19.88	0.85	0.2072
15	5	S2	EXP	3	16.7	0.87	0.2442
15	10	S2	EXP	3	16.46	0.91	0.2921
15	15	S2	EXP	3	16.58	0.90	0.3362
15	20	S2	EXP	3	16.96	0.87	0.3846
20	0	S2	EXP	3	15.93	0.91	0.2131
20	5	S2	EXP	3	13.9	0.89	0.2423
20	10	S2	EXP	3	14.21	0.93	0.2915
20	15	S2	EXP	3	14.32	0.93	0.3392
20	20	S2	EXP	3	15.19	0.91	0.3957
5	0	S2	EXP	4	20.79	0.53	0.1009
5	5	S2	EXP	4	15.79	0.69	0.1615
5	10	S2	EXP	4	15.71	0.64	0.2146
5	15	S2	EXP	4	15.69	0.58	0.2728
5	20	S2	EXP	4	15.89	0.55	0.3102
10	0	S2	EXP	4	16.75	0.84	0.1569
10	5	S2	EXP	4	13.37	0.80	0.2032
10	10	S2	EXP	4	13.29	0.83	0.2700
10	15	S2	EXP	4	13.87	0.87	0.2976
10	20	S2	EXP	4	14.15	0.87	0.3416
15	0	S2	EXP	4	14.39	0.86	0.1958
15	5	S2	EXP	4	12.46	0.86	0.2368
15	10	S2	EXP	4	12.4	0.86	0.2913
15	15	S2	EXP	4	12.31	0.86	0.3226
15	20	S2	EXP	4	12.63	0.87	0.3490
20	0	S2	EXP	4	12.14	0.88	0.2064
20	5	S2	EXP	4	10.65	0.88	0.2586
20	10	S2	EXP	4	10.54	0.86	0.2913

Table A.4: (continued)

20	15	S2	EXP	4	10.65	0.89	0.3190
20	20	S2	EXP	4	11.21	0.89	0.3628
5	0	S1	EXP	1	77.16	0.54	0.1890
5	5	S1	EXP	1	60.31	0.55	0.2957
5	10	S1	EXP	1	59.89	0.58	0.3904
5	15	S1	EXP	1	59.99	0.56	0.4804
5	20	S1	EXP	1	59.62	0.60	0.5602
10	0	S1	EXP	1	64.01	0.80	0.2464
10	5	S1	EXP	1	50.99	0.82	0.3043
10	10	S1	EXP	1	50.72	0.80	0.3736
10	15	S1	EXP	1	51.42	0.82	0.4435
10	20	S1	EXP	1	52.25	0.84	0.5129
15	0	S1	EXP	1	55.15	0.84	0.2748
15	5	S1	EXP	1	47.2	0.89	0.3221
15	10	S1	EXP	1	46.69	0.91	0.4123
15	15	S1	EXP	1	46.59	0.91	0.4430
15	20	S1	EXP	1	47.58	0.90	0.5179
20	0	S1	EXP	1	45.18	0.92	0.2964
20	5	S1	EXP	1	39.89	0.92	0.3197
20	10	S1	EXP	1	40.54	0.94	0.3754
20	15	S1	EXP	1	40.37	0.92	0.4303
20	20	S1	EXP	1	42.12	0.92	0.5142
5	0	S1	EXP	2	39.23	0.58	0.0989
5	5	S1	EXP	2	30.57	0.62	0.1481
5	10	S1	EXP	2	30.33	0.68	0.1968
5	15	S1	EXP	2	30.43	0.61	0.2380
5	20	S1	EXP	2	30.28	0.60	0.2946
10	0	S1	EXP	2	32.47	0.80	0.1371
10	5	S1	EXP	2	25.62	0.84	0.1541
10	10	S1	EXP	2	25.49	0.83	0.1921
10	15	S1	EXP	2	25.94	0.84	0.2272
10	20	S1	EXP	2	26.41	0.85	0.2754
15	0	S1	EXP	2	27.62	0.83	0.1494
15	5	S1	EXP	2	23.75	0.90	0.1683

Table A.4: (continued)

15	10	S1	EXP	2	23.6	0.91	0.1935
15	15	S1	EXP	2	23.59	0.91	0.2291
15	20	S1	EXP	2	24.05	0.89	0.2656
20	0	S1	EXP	2	23.01	0.91	0.1529
20	5	S1	EXP	2	20.35	0.90	0.1671
20	10	S1	EXP	2	20.57	0.93	0.1973
20	15	S1	EXP	2	20.52	0.92	0.2225
20	20	S1	EXP	2	21.42	0.92	0.2575
5	0	S1	EXP	3	26.2	0.61	0.0687
5	5	S1	EXP	3	20.4	0.65	0.1052
5	10	S1	EXP	3	20.3	0.64	0.1314
5	15	S1	EXP	3	20.33	0.63	0.1574
5	20	S1	EXP	3	20.23	0.65	0.1866
10	0	S1	EXP	3	22.08	0.82	0.0901
10	5	S1	EXP	3	17.5	0.87	0.1124
10	10	S1	EXP	3	17.39	0.83	0.1298
10	15	S1	EXP	3	17.62	0.85	0.1559
10	20	S1	EXP	3	17.92	0.81	0.1816
15	0	S1	EXP	3	19.3	0.84	0.1062
15	5	S1	EXP	3	16.3	0.88	0.1134
15	10	S1	EXP	3	16.34	0.91	0.1404
15	15	S1	EXP	3	16.24	0.90	0.1590
15	20	S1	EXP	3	16.46	0.86	0.1849
20	0	S1	EXP	3	15.59	0.91	0.1050
20	5	S1	EXP	3	13.78	0.89	0.1128
20	10	S1	EXP	3	14.16	0.93	0.1353
20	15	S1	EXP	3	14.02	0.91	0.1510
20	20	S1	EXP	3	14.73	0.90	0.1715
5	0	S1	EXP	4	20.18	0.60	0.0517
5	5	S1	EXP	4	15.49	0.72	0.0836
5	10	S1	EXP	4	15.49	0.65	0.1184
5	15	S1	EXP	4	15.37	0.57	0.1556
5	20	S1	EXP	4	15.33	0.54	0.1843
10	0	S1	EXP	4	16.55	0.81	0.0738

Table A.4: (continued)

10	5	S1	EXP	4	13.18	0.81	0.0911
10	10	S1	EXP	4	13.11	0.84	0.1215
10	15	S1	EXP	4	13.37	0.85	0.1530
10	20	S1	EXP	4	13.54	0.85	0.1805
15	0	S1	EXP	4	14.04	0.86	0.0880
15	5	S1	EXP	4	12.29	0.88	0.1041
15	10	S1	EXP	4	12.27	0.87	0.1338
15	15	S1	EXP	4	12.12	0.85	0.1552
15	20	S1	EXP	4	12.32	0.87	0.1837
20	0	S1	EXP	4	11.84	0.86	0.0988
20	5	S1	EXP	4	10.51	0.88	0.1089
20	10	S1	EXP	4	10.4	0.87	0.1299
20	15	S1	EXP	4	10.44	0.87	0.1501
20	20	S1	EXP	4	10.86	0.88	0.1833

Table A.2: SDD accuracy experimentation results on four-room. Run time is in seconds.

SDD Accuracy Experiment (four-room)				
Dataset	Positive	Negative	Accuracy	Run Time
STEPBALANCED	5	0	0.57	0.1865
STEPBALANCED	5	5	0.56	0.3678
STEPBALANCED	5	10	0.53	0.5455
STEPBALANCED	5	15	0.52	0.6933
STEPBALANCED	5	20	0.55	0.8511
STEPBALANCED	10	0	0.83	0.4402
STEPBALANCED	10	5	0.83	0.6784
STEPBALANCED	10	10	0.80	0.9291
STEPBALANCED	10	15	0.84	1.1165
STEPBALANCED	10	20	0.85	1.3538
STEPBALANCED	15	0	0.85	0.7088
STEPBALANCED	15	5	0.85	0.9252
STEPBALANCED	15	10	0.85	1.2187
STEPBALANCED	15	15	0.84	1.4033
STEPBALANCED	15	20	0.82	1.6698
STEPBALANCED	20	0	0.89	0.9411
STEPBALANCED	20	5	0.90	1.1590
STEPBALANCED	20	10	0.90	1.4012
STEPBALANCED	20	15	0.91	1.6837
STEPBALANCED	20	20	0.89	1.9210